HEWLETT PACKARD

HP 3000 Computer System

Machine Instruction Set

HEWLETT hp PACKARD

HP 3000
COMPUTER SYSTEMS

MACHINE INSTRUCTION SET REFERENCE MANUAL

Manual Part No. 30000-90022
Index No. 3GENL.320.30000-90022

Printed in U.S.A. 2/80

HEWLETT-PACKARD
GENERAL SYSTEMS DIVISION
19447 PRUNERIDGE AVENUE, CUPERTINO, CALIFORNIA, 95014

LIST OF EFFECTIVE PAGES

The List of Effective Pages gives the most recent date on which the technical material on any given page was altered. If a page is simply re-arranged due to a technical change on a previous page, it is not listed as a changed page. Within the manual, changes are marked with a vertical bar in the margin.

All Pages in this manual are original third edition issue.

PRINTING HISTORY

New editions incorporate all update material since the previous edition. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The date on the title page and back cover changes only when a new edition is published. If minor corrections and updates are incorporated, the manual is reprinted but neither the date on the title page and back cover nor the edition change.

First Edition .... June 1976
Second Edition .... August 1978
Third Edition .... February 1980

PREFACE

This manual contains information on the machine instruction set for the HP 3000 Computer Systems. The contents of this manual are organized as follows:

Section I contains general information on traps and interrupts, condition code, and the instruction formats used in the machine instruction description in Sections II through IV.

Section II contains information on the base instruction set furnished with the computer systems.

Section III contains information on the extended instruction set furnished with the computer systems.

Section IV contains information on the language extension instructions furnished with the computer systems.

Appendix A is an alphabetical listing of all machine instructions that gives the page on which the instruction is defined.

Except where specified, the content of this manual applies equally to all HP 3000 Computer Systems.

STACK OP INSTRUCTIONS

MNEMONIC	NAME	PAGE	MNEMONIC	NAME	PAGE
ADAX	Add A to X	2-7	FIXT	Fix and truncate	2-5
ADBX	Add B to X	2-7	FLT	Float an integer	2-3
ADD	Add A to B	2-1	FMPY	Floating point multiply	2-4
ADXA	Add X to A	2-7	FNEG	Floating point negate	2-4
ADXB	Add X to B	2-7	FSUB	Floating point subtract D,C – B,A	2-4
AND	Logical AND of A and B	2-6	INCA	Increment A	2-6
BTST	Test byte on TOS and set CC	2-9	INCB	Increment B	2-6
CAB	Rotate A-B-C	2-8	INCX	Increment X	2-6
CMP	Integer compare B, A and set CC	2-2	LADD	Logical add A + B	2-5
DADD	Double integer add D,C + B,A	2-2	LCMP	Logical compare B, A and set CC	2-5
DCMP	Double integer compare and set CC	2-3	LDIV	Logical divide C,B ÷ A	2-5
DDEL	Double delete TOS	2-9	LDXA	Load X into A	2-7
DDIV	Double integer divide	2-3	LDXB	Load X into B	2-7
DDUP	Double duplicate TOS	2-9	LMPY	Logical multiply B × A	2-5
DECA	Decrement A	2-6	LSUB	Logical subtract B – A	2-5
DECB	Decrement B	2-7	MPY	Multiply integers, integer product	2-1
DECX	Decrement X	2-6	MPYL	Multiply integers, long integer product	2-2
DEL	Delete TOS	2-9	NEG	Integer negate	2-2
DELB	Delete B	2-9	NOP	No operation	2-10
DFLT	Floating a double integer	2-3	NOT	Logical complement TOS	2-6
DIV	Integer divide B by A	2-1	OR	Logical OR of A, B	2-6
DIVL	Divide long integer C,B ÷ A	2-2	STAX	Store A into X	2-7
DMUL	Double integer multiply	2-3	STBX	Store B into X	2-7
DNEG	Double integer negate	2-3	SUB	Integer subtract B – A	2-1
DSUB	Double integer subtract D,C – B,A	2-2	TEST	Test TOS and set CC	2-9
DTST	Test double word on TOS and set CC	2-9	XAX	Exchange A and X	2-8
DUP	Duplicate TOS	2-9	XBX	Exchange B and X	2-8
DXCH	Double exchange	2-8	XCH	Exchange A and B	2-8
DZRO	Push double zero onto stack	2-8	XOR	Logical exclusive OR of A, B	2-6
FADD	Floating point add D,C + B,A	2-4	ZERO	Push integer zero onto stack	2-8
FCMP	Floating point compare and set CC	2-3	ZROB	Zero B	2-8
FDIV	Floating point divide D,C ÷ B,A	2-4	ZROX	Zero X	2-8
FIXR	Fix and round	2-4			

SHIFT INSTRUCTIONS

MNEMONIC	NAME	PAGE	MNEMONIC	NAME	PAGE
ASL	Arithmetic shift left	2-10	DLSR	Double logical shift right	2-11
ASR	Arithmetic shift right	2-10	LSL	Logical shift left	2-10
CSL	Circular shift left	2-10	LSR	Logical shift right	2-10
CSR	Circular shift right	2-11	QASL	Quadruple arithmetic shift left	2-12
DASL	Double arithmetic shift left	2-11	QASR	Quadruple arithmetic shift right	2-13
DASR	Double arithmetic shift right	2-11	TASL	Triple arithmetic shift left	2-12
DCSL	Double circular shift left	2-11	TASR	Triple arithmetic shift right	2-12
DCSR	Double circular shift right	2-12	TNSL	Triple normalizing shift left	2-12
DLSL	Double logical shift left	2-11			

FIELD AND BIT INSTRUCTIONS

MNEMONIC	NAME	PAGE	MNEMONIC	NAME	PAGE
DPF	Deposit field, A bits to B	2-14	TCBC	Test and complement bit, set CC	2-14
EXF	Extract specified field, right-justifiy	2-14	TRBC	Test and reset bit, set CC	2-13
SCAN	Scan bits	2-13	TSBC	Test and set bit, set CC	2-13
TBC	Test specified bit and set CC	2-13			

BRANCH INSTRUCTIONS

MNEMONIC	NAME	PAGE	MNEMONIC	NAME	PAGE
BCC	Branch on specified CC	2-16	BRO	Branch on TOS odd (bit 15 = 1)	2-15
BCY	Branch on carry	2-15	CPRB	Compare range and branch	2-16
BNCY	Branch on no carry	2-15	DABZ	Decrement A, branch if zero	2-15
BNOV	Branch on no overflow	2-15	DXBZ	Decrement X, branch if zero	2-14
BOV	Branch on overflow	2-15	IABZ	Increment A, branch if zero	2-14
BR	Branch unconditionally	2-16	IXBZ	Increment X, branch if zero	2-14
BRE	Branch on TOS even (bit 15 = 0)	2-15			

MOVE INSTRUCTIONS

MNEMONIC	NAME	PAGE	MNEMONIC	NAME	PAGE
CMPB	Compare bytes in two memory blocks	2-18	MVB	Move bytes in memory, addresses +/–	2-17
MABS	Move using absolute addresses	2-20	MVBL	Move words from DB+ to DL+ area	2-19
MDS	Move using data segments	2-22	MVBW	Move bytes while of specified type	2-18
MFDS	Move from data segment	2-21	MVLB	Move words from DL+ to DB+ area	2-20
MOVE	Move words in memory, address +/–	2-17	SCU	Scan bytes until test or terminal byte	2-19
MTDS	Move to data segment	2-21	SCW	Scan bytes while equal to test byte	2-19

PRIVILEGED MEMORY REFERENCE INSTRUCTIONS

MNEMONIC	NAME	PAGE	MNEMONIC	NAME	PAGE
LDEA	Load double word from extended address	2-23	PSTA	Privileged store into absolute address	2-22
LSEA	Load single word from extended address	2-23	SDEA	Store double word into extended address	2-23
LST	Load from system table	2-23	SSEA	Store single word into extended address	2-23
PLDA	Privileged load from absolute address	2-22	SST	Store into system table	2-23

IMMEDIATE INSTRUCTIONS

MNEMONIC	NAME	PAGE	MNEMONIC	NAME	PAGE
ADDI	Add immediate to integer in A	2-24	LDXI	Load X immediate	2-24
ADXI	Add immediate to X	2-25	LDXN	Load X negative immediate	2-25
ANDI	Logical AND immediate with A	2-26	MPYI	Multiply immediate with A	2-24
CMPI	Compare A with immediate, set CC	2-24	ORI	Logical OR immediate with A	2-25
CMPN	Compare A with negative immediate	2-25	SBXI	Subtract immediate from X	2-25
DIVI	Divide immediate into A	2-24	SUBI	Subtract immediate from A	2-24
LDI	Load immediate to TOS	2-24	XORI	Logical exclusive OR immediate	2-25
LDNI	Load negative immediate to TOS	2-25			

REGISTER CONTROL INSTRUCTIONS

MNEMONIC	NAME	PAGE	MNEMONIC	NAME	PAGE
ADDS	Add operand to stack pointer	2-27	SETR	Set specified registers from stack	2-26
PSHR	Push specified registers onto stack	2-26	SUBS	Subtract operand from stack pointer	2-27
RCLK	Read clock	2-27	XCHD	Exchange DB and TOS	2-27
SCLK	Store clock	2-27			

PROGRAM CONTROL AND SPECIAL INSTRUCTIONS

MNEMONIC	NAME	PAGE	MNEMONIC	NAME	PAGE
DISP	Dispatch	2-29	PSEB	Pseudo interrupt enable	2-30
EXIT	Exit from procedure	2-28	RCCR	Read system clock counter**	2-31
HALT	Halt	2-30	RSW	Read switch register	2-31
IXIT	Interrupt exit	2-29	SCAL	Subroutine call	2-28
LLBL	Load label	2-29	SCLR	Set system clock limit**	2-32
LLSH	Linked list search	2-31	SINC	Set system clock interrupt**	2-32
LOCK	Lock resource*	2-30	SXIT	Exit from subroutine	2-28
PAUS	Pause, interruptable	2-30	TOFF	Hardware timer off**	2-32
PCAL	Procedure call	2-28	TON	Hardware timer on**	2-32
PCN	Push CPU number	2-31	UNLK	Unlock resource*	2-30
PSDB	Pseudo interrupt disable	2-29	XEQ	Execute stack word	2-31
				*Series II Computer Systems only. **Series 30/33 Computer Systems only.	

I/O INSTRUCTIONS

MNEMONIC	NAME	PAGE	MNEMONIC	NAME	PAGE
CIO	Control I/O, direct*	2-34	SEML	Semaphore load**	2-37
CMD	Send command to module, direct*	2-35	SIN	Set interrupt*	2-35
DUMP	Load soft dump program**	2-35	SIO	Start I/O, block transfer*	2-33
HIOP	Hlat I/O program**	2-37	SIOP	Start I/O, program**	2-36
INIT	Initialize I/O channel**	2-36	SMSK	Set device mask	2-33
MCS	Read memory controller**	2-36	STRT	Initiate warmstart**	2-37
RIO	Read I/O, direct*	2-34	TIO	Test I/O, direct*	2-34
RIOC	Read I/O channel**	2-37	WIO	Write I/O, direct*	2-34
RMSK	Read device mask	2-33	WIOC	Write I/O, channel**	2-35
SED	Set enable/disable external interrupts	2-33		*Series II/III Computer System only. **Series 30/33 Computer System only.	

LOOP CONTROL INSTRUCTIONS

MNEMONIC	NAME	PAGE	MNEMONIC	NAME	PAGE
MTBA	Modify variable, test against limit, branch	2-38	TBA	Test variable against limit, branch	2-38
MTBX	Modify X, test against limit, branch	2-38	TBX	Test X against limit, branch	2-38

MEMORY ADDRESS INSTRUCTIONS

MNEMONIC	NAME	PAGE	MNEMONIC	NAME	PAGE
ADDM	Add memory to TOS	2-41	LDX	Load X	2-39
CMPM	Compare TOS with memory	2-41	LOAD	Load word onto stack	2-39
DECM	Decrement memory	2-41	LRA	Load relative address onto stack	2-40
INCM	Increment memory	2-41	MPYM	Multiply TOS by memory	2-41
LDB	Load byte onto stack	2-40	STB	Store byte on TOS into memory	2-40
LDD	Load double word onto stack	2-39	STD	Store double on TOS into memory	2-40
LDPN	Load double from program, negative	2-39	STOR	Store TOS into memory	2-39
LDPP	Load double from program, positive	2-39	SUBM	Subtract memory from TOS	2-41

EXTENDED INSTRUCTION SET

MNEMONIC	NAME	PAGE	MNEMONIC	NAME	PAGE
Extended-Precision Floating Point			Decimal Arithmetic		
EADD	Add	3-1	ADDD	Decimal add	3-5
ECMP	Compare	3-2	CMPD	Decimal compare	3-6
EDIV	Divide	3-2	CVAD	ASCII to decimal conversion	3-3
EMPY	Multiply	3-2	CVBD	Binary to decimal conversion	3-4
ENEG	Negate	3-2	CVDA	Decimal to ASCII conversion	3-4
ESUB	Subtract	3-1	CVDB	Decimal to binary conversion	3-5
			DMPY	Double logical multiply	3-8
			MPYD	Decimal multiply	3-8
			NSLD	Decimal normalizing left shift	3-7
			SLD	Decimal left shift	3-7
			SRD	Decimal right shift	3-7
			SUBD	Decimal subtract	3-6

LANGUAGE EXTENSION INSTRUCTIONS

MNEMONIC	NAME	PAGE	MNEMONIC	NAME	PAGE
ABSD	Absolute decimal	4-15	ENDP	End of paragraph	4-2
ABSN	Absolute numeric	4-14	LDDW	Load double word	4-16
ALGN	Align numeric	4-13	LDW	Load word	4-16
CMPS	Compare strings	4-12	NEGD	Negate decimal	4-15
CMPT	Compare translated strings	4-12	PARC	Paragraph procedure call	4-1
CVND	Convert numeric display	4-14	TR	Translate	4-11
EDIT	Edit more instructions	4-2	XBR	External branch	4-1

GENERAL INFORMATION

SECTION I

1-1. INTRODUCTION

This manual contains information on the machine instruction sets of the HP 3000 Computer Systems. Section I contains general information about the instruction sets. Section II describes each of the instructions included in the basic instruction set, Section III describes each of the instructions which are part of the HP 30012A Extended Instruction Set (EIS), and Section IV describes each of the instructions which are part of the language extension set. Appendix A contains an alphabetical listing of all instructions together with the page numbers on which the instructions are to be found.

1-2. BASIC INSTRUCTION SET

1-3. INSTRUCTION DECODING

As the CPU executes a user program, it fetches these instructions from memory. A ROM address of a microprogram stored in a microprogram ROM is generated for these instructions. There is a microprogram in ROM for each of the machine instructions. The ROM address is stored in a ROM address register (RAR). The RAR is used first to access the initial microinstruction and is then incremented to point to the next microinstruction. Thus, the entire microprogram for a particular machine instruction is called and executed by the CPU.

1-4. TRAPS AND INTERRUPTS

Only those traps and interrupts which occur as a result of instruction execution over which the user has some control are used in the instruction descriptions provided in Sections II and III. They are defined here by segment #1 segment transfer table number.

a. STT #1; BNDV — Bounds Violation. An operand or instruction is outside of the legal bounds for a particular mode of addressing.

b. STT #17; STTV — Segment Transfer Table Violation. A variety of conditions can force this trap as follows:

- The STT number in an external program label is greater than the STT length pointed to by PL in the referenced segment. This error can occur in PCAL, LLBL, and the firmware interrupt handler while attempting to set up a new segment.

- In LLBL, the label fetched from PL-N is an internal label and N is greater than 128 (%177). This would require too large an STT number when creating the external label.

- In PCAL and interrupt handler when setting up a new segment, the STT number in the external program label points to an external program label in the new segment.

- In SCAL, (PL-N) is an external label.

c. STT #18; CSTV — Code Segment Table Violation. An attempt is made to transfer to Segment 0 or 192, or a segment number is greater than the CST length.

d. STT #19; DSTV — Data Segment Table Violation. The data segment number referenced by MFDS, MTDS, or MDS is greater than the DST length or is 0.

e. STT #20; STUN — Stack Underflow. The process being executed or being transferred to is non-privileged and SM is less than DB.

f. STT #21; MODE — Privileged Mode Violation. The code segment being executed is non-privileged (bit 0 of the Status register is 0) and an attempt is made to execute a privileged instruction. This violation also occurs in EXIT if an attempt is made to exit from user to privileged mode or, if exiting from user mode, the External Interrupts bit in the Status register has been altered.

g. STT #24; STOV — Stack Overflow. SM is greater than Z or may become greater as a result of the current instruction.

h. STT #25; ARITH — Arithmetic. All User Traps will be executed in the segment #1 routine pointed to by STT #25. The error conditions and their parameters are as follows:

Interrupt Type	Octal Parameters
Integer Overflow	000001
Floating Point Overflow	000002
Floating Point Underflow	000003
Integer Divide-by-Zero	000004
Floating Point Divide-by-Zero	000005
Extended Precision FP Overflow	000010
Extended Precision FP Underflow	000011
Extended Precision FP Divide-by-Zero	000012
Decimal Overflow	000013
Invalid ASCII Digit	000014
Invalid Decimal Digit	000015
Invalid Source Word Count	000016
Result Word Count Overflow	000017
Decimal Divide-by-Zero	000020

Octal parameters 000010 through 000020 are traps for the Extended Instruction Set and are shown here for completeness only.

i. STT #31; ABS CST — Absent Code Segment. The absence bit in the CST entry for the referenced segment is set. The interrupt handler and PCAL stack a (second) marker; others including EXIT, IXIT, etc., do not.

j. STT #32; TRACE — Code Segment Trace. Code segment is being traced.

k. STT #33; UNCALL — Uncallable STT Entry. The uncallable bit in a local label or, if the STT number is 0, in (PL) is set. This trap does not stack a (second) marker.

l. STT #34; ABS DST — Absent Data Segment. The absence bit in the DST entry for the referenced segment is set.

1-5. EXTENDED INSTRUCTION SET

1-6. INSTRUCTION DECODING

Firmware in the main (basic) microprogram interprets the instructions of the extended instruction set. The operation is then like that of the basic instruction set.

1-7. INTERRUPTS

The instructions of the extended instruction set are not interruptable. If these instructions are performed by software simulation procedures, interrupts are recognized in the manner established for the instructions which make up each procedure.

1-8. EXTENDED PRECISION FLOATING POINT INSTRUCTIONS

Instruction Commentary 1 in Section III provides information on these instructions.

1-9. DECIMAL ARITHMETIC INSTRUCTION SET

Instruction Commentary 2 in Section III provides information on these instructions.

1-10. LANGUAGE EXTENSION SET

Firmware in the main (basic) microprogram interprets the instructions of the language extension set. The operation is then like that of the basic instruction set. Instruction Commentary 1, 2, and 3 in Section IV provide additional information on these instructions.

1-11. CONDITION CODE

Bits 6 and 7 of the CPU Status register are used for the Condition Code. Although several instructions make special use of the Condition Code, the Condition Code typically indicates the state of an operand (or a comparison result with two operands). The operand may be a byte, word, doubleword, tripleword, or quadrupleword, and may be located on the top of the stack, in the Index register, or in a specified memory location. Three codings are used, 00, 01, and 10. The "11" is not used. Except for the special interpretations, there are four basic patterns, table 1-1, for interpreting these codes.

Table 1-1. Condition Codes

PATTERN	CODE INTERPRETATION
Arithmetic	CCA sets CC = CCG (00) if operand greater than 0 = CCL (01) if operand less than 0 = CCE (10) if operand equals 0
Byte	CCB sets CC = CCG (00) if numeric (%060-071) = CCL (01) if special character (all others) = CCE (10) if alphabetic (%101-132 and %141-172)
Comparison	CCC sets CC = CCG (00) if operand 1 is greater than operand 2 = CCL (01) if operand 1 is less than operand 2 = CCE (10) if operand 1 equals operand 2
Direct I/O	CCD sets CC = CCG (00) if device not ready (busy) = CCL (10) if non-responding device controller = CCE (10) if responding controller and/or device ready

Pattern A is the most common Condition Code pattern. In this CCA pattern, the Condition Code is set to 00 if the operand is greater than zero, to 01 if the operand is less than zero, or to 10 if the operand is exactly zero. Since the usage of this pattern is so common, the three codes 00, 01, and 10 are named to reflect these meanings. Thus 00 is CCG ("Greater"), 01 is CCL ("Less"), and 10 is CCE ("Equal"). These names are primarily used for documentation convenience.

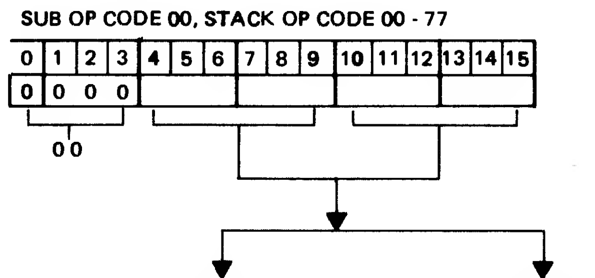
Pattern B for the Condition Code, designated CCB, is used with byte oriented instructions. In the CCB pattern, the Condition Code is set to 00 if the operand byte is an ASCII numeric character, which would be represented by octal values 060 through 071. The code is set to 10 if the byte is an ASCII alphabetic character, which would be represented by octal values 101 through 132 for upper case letters and 141 through 172 for lower case letters. The code is set to 01 if the byte is an ASCII special character represented by the remaining octal values.

Pattern C for the Condition Code, designated CCC, is used with comparison instructions. The code is set to 00 if operand 1 is greater than operand 2, or to 01 if operand 1 is less than operand 2, or to 10 if the operands are equal. In the instruction definitions, the first mentioned operand is "operand 1". For example, the definition for CMP reads: "The Condition Code is set to pattern C as a result of the integer comparison of the second word of the stack with the TOS." The second word of the stack is therefore operand 1 and the TOS is operand 2.

Pattern D for the Condition Code, designated CCD, is used with some I/O instructions. The code is set to 00 if the device is not ready. This is usually caused by the device being busy. The code is set to 01 if the device controller does not respond. Some examples of what could cause this is power off the device or controller, problems with the device or controller, or waiting for a response to an interrupt request. The last would be used with a Controller Processor. The Condition Code is set to 10 if the device and controller responded and the instruction completed normally.

1-12. INSTRUCTION FORMATS

Instruction formats are provided by figures 1-1 through 1-6.

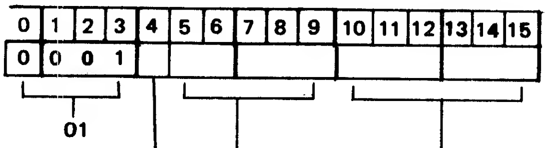


Mnemonic	Bits 4 - 9 or 10 - 15
NOP	00
DELB	01
DDEL	02
ZROX	03
INCX	04
DECX	05
ZERO	06
DZRO	07
DCMP	10
DADD	11
DSUB	12
MPYL	13
DIVL	14
DNEG	15
DXCH	16
CMP	17
ADD	20
SUB	21
MPY	22
DIV	23
NEG	24
TEST	25
STBX	26
DTST	27
DFLT	30
BTST	31
XCH	32
INCA	33
DECA	34
XAX	35
ADAX	36
ADXA	37

Mnemonic	Bits 4 - 9 or 10 - 15
DEL	40
ZROB	41
LDXB	42
STAX	43
LDXN	44
DUP	45
DDUP	46
FLT	47
FCMP	50
FADD	51
FSUB	52
FMPY	53
FDIV	54
FNEG	55
CAB	56
LCMP	57
LADD	60
LSUB	61
LMPY	62
LDIV	63
NOT	64
OR	65
XOR	66
AND	67
FIXR	70
FIXT	71
INCB	73
DECB	74
XBX	75
ADBX	76
ADXB	77

Figure 1-1. Subopcode 00 Formats

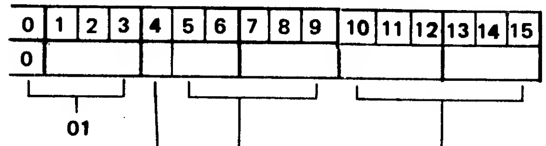
SUBOPCODE 01, OPCODES 00 - 17



Mnemonic	4	Bits 5 - 9	Bits 10 - 15
ASL	x	00	SHIFT COUNT
ASR	x	01	SHIFT COUNT
LSL	x	02	SHIFT COUNT
LSR	x	03	SHIFT COUNT
CSL	x	04	SHIFT COUNT
CSR	x	05	SHIFT COUNT
SCAN	x	06	ALL ZEROS
IABZ	I	07	± DISPLACEMENT
TASL	x	10	SHIFT COUNT
TASR	x	11	SHIFT COUNT
IXBZ	I	12	± DISPLACEMENT
DXBZ	I	13	± DISPLACEMENT
BCY	I	14	± DISPLACEMENT
BNCY	I	15	± DISPLACEMENT
TNSL	x	16	ALL ZEROS
QASL	0	17	SHIFT COUNT
QASR	1	17	SHIFT COUNT

X = INDEX BIT
I = INDIRECT BIT
SHADED BITS ARE RESERVED BITS

SUBOPCODE 01, OPCODES 20 - 37



Mnemonic	4	Bits 5 - 9	Bits 10 - 15
DASL	x	20	SHIFT COUNT
DASR	x	21	SHIFT COUNT
DLSL	x	22	SHIFT COUNT
DLSR	x	23	SHIFT COUNT
DCSL	x	24	SHIFT COUNT
DCSR	x	25	SHIFT COUNT
CPRB	I	26	± DISPLACEMENT
DABZ	I	27	± DISPLACEMENT
BOV	I	30	± DISPLACEMENT
BNOV	I	31	± DISPLACEMENT
TBC	x	32	BIT POSITION
TRBC	x	33	BIT POSITION
TSBC	x	34	BIT POSITION
TCBC	x	35	BIT POSITION
BRO	I	36	± DISPLACEMENT
BRE	I	37	± DISPLACEMENT

Figure 1-2. Subopcode 01 Formats

SUBOPCODE 02, MOVE OPCODES 00, 0 - 5

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0								

02 00

Mnemonic	Bits 8 - 10	Bits 11 - 15			
MOVE	0	B	0	0	SDEC
MVB	1	B	0	0	SDEC
MVBL	2	0	0	0	SDEC
MABS	2	0	1	SDEC	
SCW	2	1	0	0	SDEC
MTDS	2	1	1	SDEC	
MVLB	3	0	0	0	SDEC
MDS	3	0	1	SDEC	
SCU	3	1	0	0	SDEC
MFDS	3	1	1	SDEC	
MVBW	4	N	A	U	SDEC
CMPB	5	B	0	0	SDEC

Shaded bits are reserved

A = Alphabetic
B = PB/DB
N = Numeric
SDEC = S Decrement
U = Upshift

SUBOPCODE 02, MINI OPCODES 00, 14 - 17

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0								

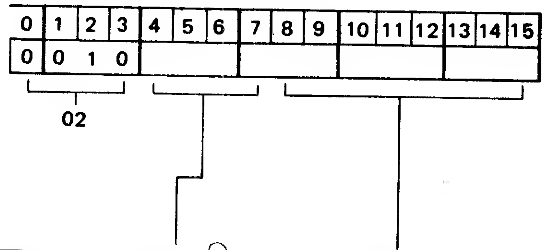
02 00

Mnemonic	Bits 8 - 11	Bits 12 - 15			
RSW	14	0	0	0	0
LLSH	14	0	0	0	1
PLDA	15	0	0	0	0
PSTA	15	0	0	0	1
LSEA	16	0	0	0	0
SSEA	16	0	0	0	1
LDEA	16	0	0	1	0
SDEA	16	0	0	1	1
IXIT	17	0	0	0	0
LOCK*	17	0	0	0	1
PCN	17	0	0	1	0
UNLK*	17	0	0	1	1

*Series II Computer Systems only.

Figure 1-3. Subopcode 02 Formats (Sheet 1 of 2)

SUBOPCODES 02, OPCODES 01 - 17



Mnemonic	Bits 4 - 7	Bits 8 - 15	
DMUL	01	CIR (8:15) = % 170	
DDIV	01	CIR (8:15) = % 171	
LDI	02	IMMEDIATE OPERAND	
LDXI	03	IMMEDIATE OPERAND	
CMPI	04	IMMEDIATE OPERAND	
ADDI	05	IMMEDIATE OPERAND	
SUBI	06	IMMEDIATE OPERAND	
MPYI	07	IMMEDIATE OPERAND	
DIVI	10	IMMEDIATE OPERAND	
PSHR	11	‡	
LDNI	12	IMMEDIATE OPERAND	
LDXN	13	IMMEDIATE OPERAND	
CMPN	14	IMMEDIATE OPERAND	
EXF	15	START BIT #	# OF BITS
DPF	16	START BIT #	# OF BITS
SETR	17	‡	

‡BIT 8 = STACK BANK REGISTER
BIT 9 = DB BANK, DB REGISTER
BIT 10 = DL REGISTER
BIT 11 = Z REGISTER
BIT 12 = STATUS REGISTER
BIT 13 = X REGISTER
BIT 14 = Q REGISTER
BIT 15 = S REGISTER

Figure 1-3 Subopcode 02 Formats (Sheet 2 of 2)

SUBOPCODE 03, SPECIAL OPCODES 00

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0								

03 (bits 0-3), 00 (bits 4-7)

Mnemonic	Bits 8 - 11	Bits 12 - 15			
LST	00	K FIELD			
PAUS	01	0	0	0	0
SED	02	0	0	0	X
XCHD	03	0	0	0	0
PSDB	03	0	0	0	1
DISP	03	0	0	1	0
PSEB	03	0	0	1	1
SMSK	04	0	0	0	0
SCLK	04	0	0	0	1
RMSK	05	0	0	0	0
RCLK	05	0	0	0	1
XEQ	06	K FIELD			
SIO	07	K FIELD			
RIO	10	K FIELD			
WIO	11	K FIELD			
TIO	12	K FIELD			
CIO	13	K FIELD			
CMD	14	K FIELD			
SST	15	K FIELD			
SIN	16	K FIELD			
HALT	17	K FIELD			

SUBOPCODE 03, OPCODES 01 - 17

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	1											

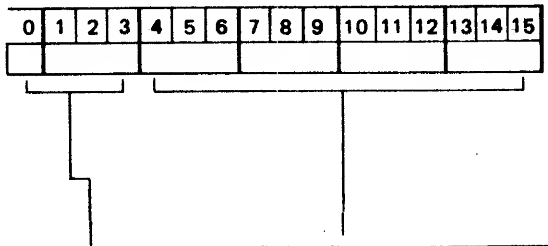
03 (bits 0-3)

Mnemonic	Bits 4 - 7	Bits 8 - 15
SCAL	01	N FIELD
PCAL	02	N FIELD
EXIT	03	N FIELD
SXIT	04	N FIELD
ADXI	05	IMMEDIATE OPERAND
SBXI	06	IMMEDIATE OPERAND
LLBL	07	PL – DISPLACEMENT
LDPP	10	P+ DISPLACEMENT
LBPN	11	P– DISPLACEMENT
ADDS	12	IMMEDIATE OPERAND
SUBS	13	IMMEDIATE OPERAND
ORI	15	IMMEDIATE OPERAND
XORI	16	IMMEDIATE OPERAND
ANDI	17	IMMEDIATE OPERAND

Shaded bits are reserved and ignored.

x = 1 or 0.

Figure 1-4. Subopcode 03 Formats



Mnemonic	Bits 0 - 3	Bits 4 - 15							
LOAD	04	X	I	MODE AND DISPLACEMENT					
TBA	05	0	0	0	±	DISPLACEMENT			
MTBA	05	0	1	0	±	DISPLACEMENT			
TBX	05	1	0	0	±	DISPLACEMENT			
MTBX	05	1	1	0	±	DISPLACEMENT			
STOR	05	x	I	1	MODE AND DISPLACEMENT				
CMPM	06	x	I	MODE AND DISPLACEMENT					
ADDM	07	x	I	MODE AND DISPLACEMENT					
SUBM	10	x	I	MODE AND DISPLACEMENT					
MPYM	11	x	I	MODE AND DISPLACEMENT					
INCM	12	x	I	0	MODE AND DISPLACEMENT				
DECM	12	x	1	I	MODE AND DISPLACEMENT				
LDX	13	x	I	MODE AND DISPLACEMENT					
BR	14	x	I	0	±	DISPLACEMENT			
BR	14	x	1	1	MODE AND DISPLACEMENT				
BCC	14	I	0	1	>	=	<	±	DISPLACEMENT
LDB	15	x	I	0	MODE AND DISPLACEMENT				
LDD	15	x	I	1	MODE AND DISPLACEMENT				
STB	16	x	I	0	MODE AND DISPLACEMENT				
STD	16	x	I	1	MODE AND DISPLACEMENT				
LRA	17	x	I	MODE AND DISPLACEMENT					

X = INDEX BIT
I = INDIRECT BIT

Figure 1-5. Subopcode 04-17 Formats

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	0	0	1			

0 2 0 4 1

Mnemonic	Bits 13 - 15
EADD	0
ESUB	1
EMPY	2
EDIV	3
ENEG	4
ECMP	5

SHADED BITS ARE RESERVED BITS
S = STACK DECREMENT
SC = SIGN CONTROL

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	1	0						

0206

Mnemonic	9	10	11	Bits 12 - 15
DMPY	0	0	0	01
CVAD	0	0	S	02
CVDA	SC		S	03
CVBD	0	0	S	04
CVDB	0	0	S	05
SLD	0	S		06
NSLD	0	S		07
SRD	0	S		10
ADDD	0	S		11
CMPD	0	S		12
SUBD	0	S		13
MPYD	0	S		14

Figure 1-6. Extended Instruction Set Formats

INSTRUCTION SET

SECTION

II

This section defines each of the machine instructions in the computer system instruction set. Where additional information would be helpful in understanding the operation of a particular instruction, an instruction commentary reference is given following the definition. In such cases, refer to the corresponding number under the heading, "Instruction Commentary", at the end of this section.

STACK OP INSTRUCTIONS

INTEGER INSTRUCTIONS

ADD Add. The top two words of the stack are added in integer form and are then deleted. The resulting sum is pushed onto the stack.
Stack opcode: 20
Indicators: CCA, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	0	0	0	0						

Alternate Position

SUB Subtract. The TOS is subtracted in integer form from the second word of the stack and both words are then deleted. The resulting difference is then pushed onto the stack.
Stack opcode: 21
Indicators: CCA, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	0	0	0	1						

Alternate Position

MPY Multiply. The top two words of the stack are multiplied in integer form. The two words are deleted and the least significant word of the double length product is pushed onto the stack. If the high order 17 bits of the double length product (including the sign bit of the second word) are not all zeros or all ones, Overflow is set.
Instruction Commentary 1.
Stack opcode: 22
Indicators: CCA, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	0	0	1	0						

Alternate Position

DIV Divide. The integer in the second word of the stack is divided by the integer on the TOS. The second word is replaced by the quotient, and the top word is replaced by the remainder.
Stack opcode: 23
Indicators: CCA on quotient, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	0	0	1	1						

Alternate Position

NEG Negate. The integer in the TOS is replaced by its two's complement.
Stack opcode: 24
Indicators: CCA, Overflow, Carry
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	0	1	0	0						

Alternate Position

CMP Compare. The Condition Code is set to pattern C as a result of the integer comparison of the second word of the stack with the TOS. Both words are deleted.
Stack opcode: 17
Indicators: CCC
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	1	1	1	1						

Alternate Position

DOUBLE INTEGER INSTRUCTIONS

DADD Double add. The two doubleword integers contained in the top four elements of the stack are added in double length integer form (D,C + B,A) and they are deleted. The doubleword integer sum is pushed onto the stack (B,A).
Stack opcode: 11
Indicators: CCA, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	1	0	0	1						

Alternate Position

DSUB Double subtract. The doubleword integer contained in the top two words of the stack is subtracted from the doubleword integer contained in the third and fourth words of the stack (D,C − B,A). The top four words of the stack are deleted and the doubleword integer result is pushed onto the stack (B,A).
Stack opcode: 12
Indicators: CCA, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	1	0	1	0						

Alternate Position

MPYL Multiply long. The top two words of the stack are multiplied in integer form. The words are replaced by the double length product, with the least significant half on the TOS. Overflow is cleared. Carry is cleared if the low order 16 bits represent the true result (i.e., if the high order 17 bits are either all zeros or all ones); otherwise, Carry is set.
Instruction Complementary 1.
Stack opcode: 13
Indicators: CCA, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	1	0	1	1						

Alternate Position

DIVL Divide long. The doubleword integer in the second and third elements of the stack is divided by the integer in the TOS (C,B ÷ A). The three words are deleted, and the quotient and remainder are pushed onto the stack (quotient in B, remainder in A).
Stack opcode: 14
Indicators: CCA, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	1	1	0	0						

Alternate Position

DNEG Double negate. The doubleword integer contained in the top two words of the stack is negated (two's complemented) and replaces the original doubleword integer.
Stack opcode: 15
Indicators: CCA, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	1	1	0	1						

Alternate Position (bits 10–15)

DCMP Double compare. The Condition Code is set to pattern C as a result of the doubleword integer comparison of D,C and B,A. The two double words are deleted from the stack.
Stack opcode: 10
Indicators: CCC
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	1	0	0	0						

Alternate Position (bits 10–15)

DMUL Double integer multiply. The two's complement double integer contained in D and C is multiplied by the two's complement double integer contained in B and A. The four words are popped from the stack and the least significant doubleword of the product is pushed onto the stack. If the high order 33 bits if the 64-bit product are not all zeros or all ones, overflow is set.
Sub-opcode 2: 01, bits 8-15 = 170
Indicators: CCA
Traps. STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	1	1	1	1	0	0	0

CIR 8:8 (bits 8–15)

DDIV Double integer divide. The two's complement double integer contained in D and C is divided by the two's complement double integer contained in B and A. The four words are popped from the stack, the 32-bit quotient is pushed into D and C, and the 32-bit remainder is pushed into B and A.
Sub-opcode 2: 01, bits 8-15 = 171
Indicators: CCA on quotient Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	1	1	1	1	0	0	1

CIR 8:8 (bits 8–15)

FLOATING POINT INSTRUCTIONS

DFLT Double float. Converts the doubleword integer contained in the top two words of the stack to a floating point number with rounding.
Instruction Commentary 2.
Stack opcode: 30
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	0	0	0						

Alternate Position (bits 10–15)

FLT Float. Converts the integer on the TOS to a 32-bit floating point number with rounding. The TOS is deleted and the doubleword floating point result is pushed onto the stack.
Instruction Commentary 2.
Stack opcode: 47
Indicators: CCA
Traps: STUN, STOV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	1	1	1						

Alternate Position (bits 10–15)

FCMP Floating compare. The Condition Code is set to pattern C as a result of the floating point comparison of D,C with B,A. The two floating point double words are deleted.
Stack opcode: 50
Indicators: CCC
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	1	0	0	0						

Alternate Position (bits 10–15)

FADD Floating add. The two floating point numbers contained in the top four words of the stack are added in floating point form. The top four words of the stack are deleted and the two-word sum is pushed onto the stack.
Instruction Commentary 2.
Stack opcode: 51
Indicators: CCA, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	1	0	0	1						

Alternate Position

FSUB Floating subtract. The floating point number contained in the top two words of the stack is subtracted in floating point form from the floating point number contained in the third and fourth words of the stack. The top four words of the stack are deleted and the two-word difference is pushed onto the stack.
Instruction Commentary 2.
Stack opcode: 52
Indicators: CCA, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	1	0	1	0						

Alternate Position

FMPY Floating multiply. The two floating point numbers contained in the top four words of the stack are multiplied in floating point form. The top four words of the stack are deleted and the two-word result is pushed onto the stack.
Instruction Commentary 2.
Stack opcode: 53
Indicators: CCA, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	1	0	1	1						

Alternate Position

FDIV Floating divide. The floating point number contained in the third and fourth words of the stack is divided by the floating point number contained in the top two words of the stack. The top four words of the stack are deleted and the two-word quotient is pushed onto the stack.
Instruction Commentary 2.
Stack opcode: 54
Indicators: CCA, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	1	1	0	0						

Alternate Position

FNEG Floating negate. The floating point number contained in the top two words of the stack is negated in floating point form.
Stack opcode: 55
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	1	1	0	1						

Alternate Position

FIXR Fix and round. The floating point number contained in the top two words of the stack is converted to fixed point form and rounded to the nearest double word integer. Carry is cleared if the low order 16 bits of the double word result (TOS) represent the true integer value (i.e., if the high order 17 bits are either all zeros or all ones); otherwise Carry is set.
Instruction Commentaries 1 and 2.
Stack opcode: 70
Indicators: CCA, Carry, Overflow
Traps: STUN, ARITH

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	0	0	0						

Alternate Position

FIXT Fix and truncate. The floating point number contained in the top two words of the stack is converted to fixed point form and truncated to a double word integer. Carry is cleared if the low order 16 bits of the double word result (TOS) represent the true integer value (i.e., if the high order 17 bits are either all zeros or all ones); otherwise Carry is set.
Instruction Commentaries 1 and 2.
Stack opcode: 71
Indicators: CCA, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	0	0	1						

Alternate Position

LOGICAL INSTRUCTIONS

LCMP Logical compare. The Condition Code is set to pattern C as a result of the comparison of the second word of the stack with the TOS. The two words are then deleted from the stack.
Stack opcode: 57
Indicators: CCC
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	1	1	1	1						

Alternate Position

LADD Logical add. The top two words of the stack are added as 16-bit positive integers, and they are deleted from the stack. The resulting sum is pushed onto the stack.
Stack opcode: 60
Indicators: CCA (as a 2's complement result), Carry
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	0	0	0	0						

Alternate Position

LSUB Logical subtract. The top word of the stack is subtracted in logical form from the second word and they are deleted. The resulting difference is pushed onto the stack.
Stack opcode: 61
Indicators: CCA (as a 2's complement result), Carry
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	0	0	0	1						

Alternate Position

LMPY Logical multiply. The top two words of the stack are multiplied as 16-bit positive integers. The words are replaced by the double length product with the least significant half on the TOS. Carry is cleared if the TOS word of the result represents the true integer value (i.e., if the high order 16 bits are all zeros); otherwise, Carry is set.
Instruction Commentary 1.
Stack opcode: 62
Indicators: CCA (as a 2's complement result), Carry
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	0	0	1	0						

Alternate Position

LDIV Logical divide. The 32-bit positive integer in the second and third words of the stack is divided by the 16-bit positive integer on the TOS (C,B ÷ A). The top three words are deleted. The quotient is pushed onto the stack (B) and then the remainder (A). If overflow occurs, the quotient will be modulo 2^{16}.
Stack opcode: 63
Indicators: CCA on quotient (as a 2's complement result), Overflow
Trap: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	0	0	1	1						

Alternate Position

NOT — One's complement. The top word of the stack is converted to its one's complement.
Stack opcode: 64
Indicators: CCA
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	0	1	0	0						

Alternate Position

BOOLEAN INSTRUCTIONS

OR — Logical OR. The top two words of the stack are merged by a logical inclusive-OR. The two words are deleted and the result is pushed onto the stack.
Stack opcode: 65
Indicators: CCA on the new TOS
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	0	1	0	1						

Alternate Position

XOR — Logical exclusive-OR. The top two words of the stack are combined by a logical exclusive-OR. The two words are deleted and the result is pushed onto the stack.
Stack opcode: 66
Indicators: CCA on the new TOS
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	0	1	1	0						

Alternate Position

AND — Logical AND. The top two words of the stack are combined by a logical AND. The two words are deleted and the result is pushed onto the stack.
Stack opcode: 67
Indicators: CCA on the new TOS
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	0	1	1	1						

Alternate Position

INCREMENT/DECREMENT INSTRUCTIONS

INCX — Increment X. The content of the Index register is incremented by one in integer form.
Stack opcode: 04
Indicators: CCA, Carry, Overflow
Traps: ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	1	0	0						

Alternate Position

DECX — Decrement X. The content of the Index register is decremented by one in integer form.
Stack opcode: 05
Indicators: CCA, Carry, Overflow
Traps: ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	1	0	1						

Alternate Position

INCA — Increment A. The TOS is incremented by one in integer form.
Stack opcode: 33
Indicators: CCA, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	0	1	1						

Alternate Position

DECA — Decrement A. The TOS is decremented by one in integer form.
Stack opcode: 34
Indicators: CCA, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	1	0	0						

Alternate Position

INCB — Increment B. The second word of the stack is incremented by one in integer form. The TOS is unaffected.
Stack opcode: 73
Indicators: CCA, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	0	1	1						

Alternate Position

DECB Decrement B. The second word of the stack is decremented by one in integer form. The TOS is unaffected.
Stack opcode: 74
Indicators: CCA, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	1	0	0						

Alternate Position (bits 10–15)

INDEX INSTRUCTIONS

STBX Store B into X. The second word of the stack replaces the content of the Index register.
Stack opcode: 26
Indicators: CCA, on the new X
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	0	1	1	0						

Alternate Position

ADAX Add A to X. The TOS is added in integer form to the content of the Index register. The sum replaces the content of the Index register, and the TOS is deleted.
Stack opcode: 36
Indicators: CCA on the new X, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	1	1	0						

Alternate Position

ADXA Add X to A. The content of the Index register is added to the TOS, and the sum replaces the TOS.
Stack opcode: 37
Indicators: CCA on the new TOS, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	1	1	1						

Alternate Position

LDXB Load X into B. The second word of the stack is replaced by the content of the Index register. The TOS is unaffected.
Stack opcode: 42
Indicators: CCA on the new B
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	0	1	0						

Alternate Position

STAX Store A into X. The TOS replaces the content of the Index register, and TOS is deleted from the stack.
Stack opcode: 43
Indicators: CCA on the new X
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	0	1	1						

Alternate Position

LDXA Load X onto stack. The content of the Index register is pushed onto the stack.
Stack opcode: 44
Indicators: CCA on to the new TOS
Traps: STOV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	1	0	0						

Alternate Position

ADBX Add B to X. The second word of the stack is added in integer form to the content of the Index register, and the result replaces the content of the Index register.
Stack opcode: 76
Indicators: CCA on the new X, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	1	1	0						

Alternate Position

ADXB Add X to B. The content of the Index register is added in integer form to the second word of the stack, and the sum replaces the second word of the stack.
Stack opcode: 77
Indicators: CCA on the new B, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	1	1	1						

Alternate Position

EXCHANGE INSTRUCTIONS

DXCH **Double exchange. The top two doubleword pairs are interchanged on the stack.**
Stack opcode: 16
Indicators: CCA on the new TOS double word
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	1	1	1	0						

Alternate Position

XCH **Exchange A and B. The top two words of the stack are interchanged.**
Stack opcode: 32
Indicators: CCA on the new TOS
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	0	1	0						

Alternate Position

XAX **Exchange A and X. The content of the TOS and the Index register are interchanged.**
Stack opcode: 35
Indicators: CCA on the new TOS
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	1	0	1						

Alternate Position

CAB **Rotate A,B,C. The third word of the stack is removed from the stack, the two top words are compressed onto the rest of the stack, and the original third word is pushed onto the stack.**
Stack opcode: 56
Indicators: CCA on the new TOS
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	1	1	1	0						

Alternate Position

XBX **Exchange B and X. The second word of the stack is interchanged with the content of the Index register.**
Stack opcode: 75
Indicators: unaffected
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	1	1	1	0	1						

Alternate Position

ZERO INSTRUCTIONS

ZROX **Zero X. The content of the Index register is replaced by zero.**
Stack opcode: 03
Indicators: unaffected
Traps: None

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	1	1						

Alternate Position

'ERO **Push zero. A zero word is pushed onto the stack.**
Stack opcode: 06
Indicators: unaffected
Traps: STOV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	1	1	0						

Alternate Position

'RO **Push double zero. Two words containing all zeros are pushed onto the stack.**
Stack opcode: 07
Indicators: unaffected
Traps: STOV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	1	1	1						

Alternate Position

ZROB **Zero B. The second word of the stack is replaced by zero. The TOS is unaffected.**
Stack opcode: 41
Indicators: unaffected
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	0	0	1						

Alternate Position

DUPLICATE AND DELETE INSTRUCTIONS

DELB Delete B. The second word of the stack is deleted and the stack is compressed. The content of the TOS is unchanged.
Stack opcode: 01
Indicators: unaffected
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	1						

Alternate Position

DDEL Double delete. The top two words of the stack are deleted.
Stack opcode: 02
Indicators: unaffected
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	1	0						

Alternate Position

DEL Delete A. The top word of the stack is deleted.
Stack opcode: 40
Indicators: unaffected
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	0	0	0						

Alternate Position

DUP Duplicate A. The top word of the stack is duplicated by pushing a copy of the TOS onto the stack.
Stack opcode: 45
Indicators: CCA
Traps: STUN, STOV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	1	0	1						

Alternate Position

DDUP Double duplicate. The double word in the top two words of the stack is duplicated by pushing a copy of it onto the stack.
Stack opcode: 46
Indicators: CCA on new TOS double word
Traps: STUN, STOV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	0	0	1	1	0						

Alternate Position

TEST INSTRUCTIONS

TEST Test TOS. The condition code is set to pattern A according to the content of the TOS word.
Stack opcode: 25
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	0	1	0	1						

Alternate Position

DTST Test double word on TOS. The condition code is set to pattern A according to the contents of the top two words of the stack. Also, Carry is cleared if the low order 16 bits of the doubleword result (TOS) represent the true integer value (i.e., if the high order 17 bits are either all zeros or all ones); otherwise, Carry is set.
Instruction Commentary 1.
Stack opcode: 27
Indicators: CCA, Carry
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	0	1	1	1						

Alternate Position

BTST Test byte on TOS. The Condition Code is set to pattern B according to the contents of the byte contained in the eight least significant bits of the TOS word (bits 8-15).
Stack opcode: 31
Indicators: CCB
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	1	1	0	0	1						

Alternate Position

NO OP INSTRUCTION

NOP No operation. The user's program space and data space remain unchanged.
Stack opcode: 00
Indicators: unaffected
Traps: None

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0						
										Alternate Position					

SHIFT INSTRUCTIONS

SINGLE WORD SHIFT INSTRUCTIONS

All single word shift instructions: Instruction Commentary 3.

ASL Arithmetic shift left. The TOS is shifted left n bits, preserving the sign bit. The value of n (modulo 64) is the number specified in the argument field plus, if X is specified (bit 4), the content of the Index register.
Sub-opcode 1: 00
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	0	0	0	0	0						
										Shift Count					

ASR Arithmetic shift right. The TOS is shifted right n places, propagating the sign bit. The value of n (modulo 64) is the number specified in the argument field plus, if X is specified, the content of the Index register.
Sub-opcode 1: 01
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	0	0	0	0	1						
										Shift Count					

LSL Logical shift left. The TOS is shifted left n bits logically. The value of n (modulo 64) is the number specified in the argument field plus, if X is specified, the content of the Index register.
Sub-opcode 1: 02
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	0	0	0	1	0						
										Shift Count					

LSR Logical shift right. The TOS is shifted right n bits logically. The value of n (modulo 64) is the number specified in the argument field plus, if X is specified, the content of the Index register.
Sub-opcode 1: 03
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	0	0	0	1	1						
										Shift Count					

CSL Circular shift left. The TOS is shifted left n bits circularly. The value of n (modulo 64) is the number specified in the argument field plus, if X is specified, the content of the Index register.
Sub-opcode 1: 04
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	0	0	1	0	0						
										Shift Count					

CSR Circular shift right. The TOS is shifted right n bits circularly. The value of n (modulo 64) is the number specified in the argument field plus, if X is specified, the content of the Index register.
Sub-opcode 1: 04
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	0	0	1	0	1	Shift Count					

DOUBLE WORD SHIFT INSTRUCTIONS

All double word shift instructions: Instruction Commentaries 3 and 4.

DASL Double arithmetic shift left. The double word contained in the top two words of the stack is shifted left n bits, preserving the sign bit (bit 0 of B). The value of n (modulo 64) is the number specified in the argument field plus, if X is specified, the content of the Index register.
Sub-opcode 1: 20
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	1	0	0	0	0	Shift Count					

DASR Double arithmetic shift right. The double word contained in the top two words of the stack is shifted right n bits, propagating the sign bit (bit 0 of B). The value of n (modulo 64) is the number specified in the argument field plus, if X is specified, the content of the Index register.
Sub-opcode 1: 21
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	1	0	0	0	1	Shift Count					

DLSL Double logical shift left. The double word contained in the top two words of the stack is shifted left n bits logically. The value of n (modulo 64) is the number specified in the argument field plus, if X is specified, the content of the Index register.
Sub-opcode 1: 22
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	1	0	0	1	0	Shift Count					

DLSR Double logical shift right. The double word contained in the top two words of the stack is shifted right n bits logically. The value of n (modulo 64) is the number specified in the argument field plus, if X is specified, the content of the Index register.
Sub-opcode 1: 23
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	1	0	0	1	1	Shift Count					

DCSL Double circular shift left. The double word contained in the top two words of the stack is shifted left n bits circularly. The value of n (modulo 64) is the number specified in the argument field plus, if X is specified, the content of the Index register.
Sub-opcode 1: 24
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	1	0	1	0	0	Shift Count					

DCSR Double circular shift right. The double word contained in the top two words of the stack is shifted right n bits circularly. The value of n (modulo 64) is the number specified in the argument field plus, if X is specified, the content of the Index register.
Sub-opcode 1: 25
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	1	0	1	0	1	Shift Count					

TRIPLE WORD SHIFT INSTRUCTIONS

All triple word shift instructions: Instruction Commentaries 3 and 5.

TASL Triple arithmetic shift left. The triple word integer contained in the top three words of the stack is shifted left n bits, preserving the sign bit (bit 0 of C). The value of n (modulo 64) is the number specified in the argument field plus, if X is specified, the content of the Index register.
Sub-opcode 1: 10
Indicators: CCA on the new TOS triple word
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	0	1	0	0	0	Shift Count					

TASR Triple arithmetic shift right. The triple word integer contained in the top three words of the stack is shifted right n bits, propagating the sign bit (bit 0 of C). The value of n (modulo 64) is the number specified in the argument field plus, if X is specified, the content of the Index register.
Sub-opcode 1: 11
Indicators: CCA on the new TOS triple word
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	0	1	0	0	1	Shift Count					

TNSL Triple normalizing shift left. The top three words of the stack are shifted left arithmetically until bit 6 of C is a "1". Bits 0 through 5 of C are cleared ("0"). The shift count is stored in the Index register. The instruction initially clears the Index register unless X is specified ("1" in bit 4 of the instruction).
Sub-opcode 1: 16
Indicators: CCA on final value of top three words
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	0	1	1	1	0	0	0	0	0	0	0
										Reserved					

QUADRUPLE WORD SHIFT INSTRUCTIONS

Quadruple word shift instructions: Instruction Commentaries 3 and 6.

QASL Quadruple arithmetic shift left. The four-word integer contained in the top four words of the stack is shifted left n bits, preserving the sign (bit 0 of word D). The value of n (modulo 64) is the number specified in the shift count plus the contents of the Index register.
Sub-opcode 1: 17, bit 4 = 0
Indicators: CCA on the new TOS quadruple word
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	0	0	1	1	1	1	Shift Count					

QASR Quadruple arithmetic shift right. The four-word integer contained in the top four words of the stack is shifted right n bits, preserving the sign (bit 0 of word D). The value of n (modulo 64) is the number specified in the shift count plus the contents of the Index register.
Sub-opcode 1: 17, bit 4 = 1
Indicators: CCA on the new TOS quadruple word
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	1	0	1	1	1	1	Shift Count					

FIELD AND BIT INSTRUCTIONS

SCAN Scan bits. The TOS is shifted left until bit 0 contains a "1", then is shifted left one more bit. The shift count is left in the Index register, indicating the bit position which contained the "1". The instruction normally sets the Index register to −1 before beginning the shifts. However, if X is specified, the shift count adds on to the existing Index register content. If TOS is all zeros, the count will be 16 if unindexed, or X + 16 if indexed.
Sub-opcode 1: 06
Indicators: CCA on final TOS
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	0	0	1	1	0	0	0	0	0	0	0
										Reserved					

TBC Test bit and set Condition Code. One bit of the TOS word is tested and the Condition Code is set to a special pattern depending on the state of the bit. The bit position to be tested is specified by the argument field of the instruction plus, if X is specified, the content of the Index register. If the number specified exceeds 15, the bit position indicated is modulo 16; e.g., bit 0 is tested for counts of 0, 16, 32, 48, etc.
Sub-opcode 1: 32
Indicators: CCE if the bit was "0"
CCL or CCG if the bit was "1"
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	1	1	0	1	0	Bit Position					

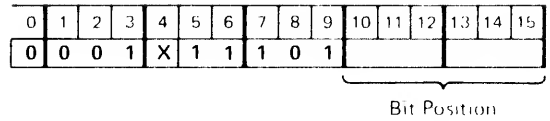
TRBC Test and reset bit, set Condition Code. The operation of this instruction is identical to that of TBC except that the tested bit is reset to "0" after the test.
Sub-opcode 1: 33
Indicators: CCE if the bit was "0"
CCL or CCG if the bit was "1"
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	1	1	0	1	1	Bit Position					

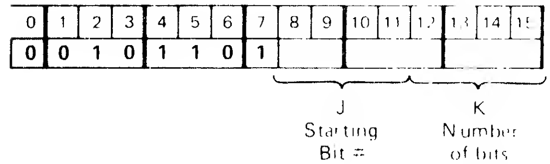
TSBC Test and set bit, set Condition Code. The operation of this instruction is identical to that of TBC except that the tested bit is set to "1" after the test.
Sub-opcode 1: 34
Indicators: CCE if the bit was "0"
CCL or CCG if the bit was "1"
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	X	1	1	1	0	0	Bit Position					

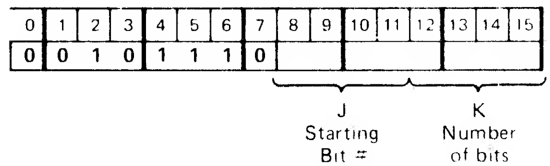
TCBC Test and complement bit, set Condition Code. The operation of this instruction is identical to that of TBC except that the tested bit is complemented after the test.
Sub-opcode 1: 35
Indicators: CCE if the bit was "0"
CCL or CCG if the bit was "1"
Traps: STUN



EXF Extract field. A specified set of bits in the TOS are extracted and right justified, and the result, with high order zeros, replaces the TOS. The J field specifies the starting (leftmost) bit number in the source field, and the K field specifies the number of bits to be extracted.
Instruction Commentary 7
Sub-opcode 2: 15
Indicators: CCA on the new TOS
Traps: STUN

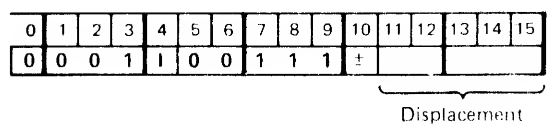


DPF Deposit field. A specified number of the least significant bits of the TOS are deposited in the second word of the stack, beginning at the bit number specified by the J field; the remaining bits of the second word of the stack are unchanged. The K field specifies the number of bits to be deposited. The source operand is deleted from the stack.
Instruction Commentary 7
Sub-opcode 2: 16
Indicators: CCA on the new TOS
Traps: STUN

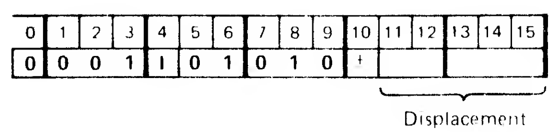


BRANCH INSTRUCTIONS

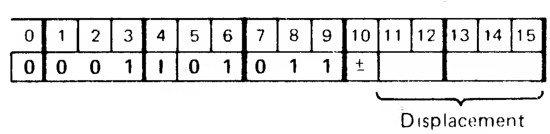
IABZ Increment A, branch if zero. The TOS is incremented. If the result is then zero, control is transferred to P ± displacement; otherwise to P+1.
Sub-opcode 1: 07
Indicators: CCA, Carry, Overflow
Addressing modes: P relative (+/−)
Direct or indirect
Traps: STUN, BNDV if user or privileged, ARITH



IXBZ Increment X, branch if zero. The Index register is incremented if the result is then zero, control is transferred to P ± displacement; otherwise to P+1.
Sub-opcode 1: 12
Indicators: CCA, Carry, Overflow
Addressing modes: P relative (+/−).
Direct or indirect
Traps: BNDV if user or privileged, ARITH



DXBZ Decrement X, branch if zero, The Index register is decremented. If the result is then zero, control is transferred to P ± displacement; otherwise to P+1.
Sub-opcode 1: 13
Indicators: CCA, Carry, Overflow
Addressing modes: P relative (+/−)
Direct or indirect
Traps: BNDV if user or privileged, ARITH



DABZ Decrement A, branch if zero. The TOS is decremented. If the result is then zero, control is transferred to P ± displacement; otherwise to P+1.
Sub-opcode 1: 27
Indicators: CCA, Carry, Overflow
Addressing modes: P relative (+/–)
Direct or indirect
Traps: STUN, BNDV if user or privileged, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	1	0	1	1	1	±					

Displacement

BCY Branch on carry. If the Carry bit of the Status register is set ("1"), control is transferred to P ± displacement; otherwise to P+1.
Sub-opcode 1: 14
Indicators: Carry cleared
Addressing modes: P relative (+/–)
Direct or indirect
Traps: BNDV if user or privileged

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	0	1	1	0	0	±					

Displacement

BNCY Branch on no carry. If the Carry bit of the Status register is clear ("0"), control is transferred to P ± displacement; otherwise to P+1.
Sub-opcode 1: 15
Indicators: Carry cleared
Addressing modes: P relative (+/–)
Direct or indirect
Traps: BNDV if user or privileged

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	0	1	1	0	1	±					

Displacement

BOV Branch on overflow. If the Overflow bit of the Status register is set ("1"), control is transferred to P± displacement; otherwise to P+1.
Sub-opcode 1: 30
Indicators: Overflow cleared
Addressing modes: P relative (+/–)
Direct or indirect
Traps: BNDV if user or privileged

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	1	1	0	0	0	±					

Displacement

BNOV Branch on no overflow. If the Overflow bit of the Status register is clear ("0"), control is transferred to P ± displacement; otherwise to P+1.
Sub-opcode 1: 31
Indicators: Overflow cleared
Addressing modes: P relative (+/–)
Direct or indirect
Traps: BNDV if user or privileged

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	1	1	0	0	1	±					

Displacement

BRO Branch on TOS odd. If the TOS is odd (bit 15 = 1), control is transferred to P ± displacement; otherwise to P+1. The TOS is deleted.
Sub-opcode 1: 36
Indicators: unaffected
Addressing modes: P relative (+/–)
Direct or indirect
Traps: STUN, BNDV if user or privileged

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	1	1	1	1	0	±					

Displacement

BRE Branch to TOS even. If the TOS is even (bit 15 = 0), control is transferred to P ± displacement; otherwise to P+1. The TOS is deleted.
Sub-opcode 1: 37
Indicators: unaffected
Addressing modes: P relative (+/–)
Direct or indirect
Traps: STUN, BNDV if user or privileged

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	1	1	1	1	1	±					

Displacement

CPRB Compare range and branch. The integer in the Index register is tested to determine if it is within the interval defined by the upper bound integer on the TOS and the lower bound integer in the second word of the stack. The Condition Code is set by the comparison to a special pattern: CCE if within range, CCL if below range, CCG if above range. If the integer in the Index register is within the specified range, control is then transferred to P ± displacement; otherwise to P+ 1. The top two elements of the stack are deleted in either case.

Sub-opcode 1: 26
Indicators: CCE, CCL, CCG
Addressing modes: P relative (+/–)
Direct or indirect
Traps: STUN, BNDV if user or privileged

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	I	1	0	1	1	0	±					

Displacement

BR Branch unconditionally. For P relative mode, control is transferred unconditionally to P ± displacement, plus (if specified) the value in X; may be indirect. For DB, Q, and S relative modes, control is transferred indirectly (only) via the location specified by DB, Q, or S ± this displacement; the content of the location so specified is added to PB (plus post-indexing if X is specified) to obtain the effective address for P.

Instruction Commentary 8
Memory opcode: 14, bits 5, 6 = 00, 10, or 11
Indicators: unaffected
Addressing modes: P relative (+/–), direct or indirect
DB+ relative, indirect
Q+ relative, indirect
Q– relative, indirect
S– relative, indirect
Indexing available
Traps: BNDV, BNDV on P and P relative if user or privileged

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	0	X	I	0	±								

Displacement
P Relative

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	0	X	1	1									

				Displacement
DB+	0			
Q+	1	0		
Q-	1	1	0	
S-	1	1	1	

BCC Branch on Condition Code. The Condition Code in the Status register is compared with conditions named in the CCF field of the instruction. If the named conditions are met, control is transferred to P ± displacement; otherwise to P+ 1. The displacement is limited to ± 31. Control is transferred to the branch address under the following conditions:

If CCF	=	0, never branch		
	=	1, branch if CC	=	CCL
	=	2, branch if CC	=	CCE
	=	3, branch if CC	=	CCL or CCE
	=	4, branch if CC	=	CCG
	=	5, branch if CC	=	CCG or CCL
	=	6, branch if CC	=	CCG or CCE
	=	7, always branch		

Memory opcode: 14, bits 5,6 = 01
Indicators: unaffected
Addressing modes: P relative (+/–)
Direct or indirect
Traps: BNDV if user or privileged

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	0	I	0	1	G	E	L	±					

CCF Displacement

MOVE INSTRUCTIONS

NOTE

All Move instructions are interruptable after each word (or byte) transfer and will continue from the point of interrupt when control is returned to the instruction.

MOVE Move words. This instruction transfers a specified number of words from one area of primary memory to another. The instruction expects a signed word count in A, a DB or PB relative displacement for a source address in B, and a DB relative displacement for a target address in C. As long as the word count in A has not been counted to zero, the transferring of data will continue as follows: The content of the memory location specified by DB + B or PB + B is transferred to the location specified by DB + C. If the word count in A is positive, the source and target displacement values in B and C are incremented by one on each transfer, and the word count is decremented by one. If the word count in A is negative, the source and target displacement values in B and C are decremented by one on each transfer, and the word count is incremented by one. Note that the word count is always changed by one toward zero. On completion of the block transfer, the instruction deletes from the stack the number of words specified by the SDEC (S decrement) field of the instruction; the range of this field is 0 through 3.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	0	0	0		0	0		
											PB/DB			SDEC	

Instruction Commentary 9
Move opcode: 0
Indicators: unaffected
Addressing modes: DB+ or PB+ for source
DB+ for target
Traps: STUN, STOV, BNDV, BNDV on P relative if user or privileged

MVB Move bytes. The MVB instruction transfers a specified number of bytes from one area of primary memory to another. The instruction expects a signed byte count in A, a DB or PB relative displacement for a source byte address in B, and a DB relative displacement for a target byte address in C. As long as the word count in A has not been counted to zero, the transferring of data will continue as follows: The content of the byte address location specified by DB + B or PB + B is transferred to the byte address location specified by DB + C. If the byte count in A is positive, the source and target displacement values in B and C are incremented by one on each transfer, and the byte count is decremented by one. If the byte count in A is negative, the source and target displacement values in B and C are decremented by one on each transfer, and the byte count is incremented by one. Note that the byte count is always changed by one toward zero. On completion of the block transfer, the instruction deletes from the stack the number of words (0, 1, 2, or 3) specified by the SDEC field of the instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	0	0	1		0	0		
											PB/DB			SDEC	

Instruction Commentary 9
Move opcode: 1
Indicators: unaffected
Addressing modes: Byte addressing
DB+ or PB+ for source
DB+ for target
Traps: STUN, STOV, BNDV, BNDV on P relative if user or privileged

MVBW Move bytes while of specified type. This instruction transfers an unspecified number of bytes from one area of primary memory to another. The instruction expects a source byte address in the TOS and a DB relative displacement for a target byte address in the second word of the stack. As long as the source byte is of the type specified in the CCF field, it is moved to the target area. The target displacement value in B is incremented by one on each transfer. If the byte to be moved is a lower case letter and the upshift bit is on, the target byte will be an upshifted copy of the source byte. Byte transfers continue until the source byte is not of the proper type. On completion of the block transfer, the instruction deletes from the stack the number of words (0, 1, 2, or 3) specified by the SDEC field of the instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	0	0					

CCF (bits 11–12); Upshift (bit 13); SDEC (bits 14–15)

	CCF
Alphabetic:	0 1
Numeric:	1 0

Instruction Commentary 9
Move opcode: 4
Indicators: CCB on the last character scanned
Addressing mode: Byte addressing, DB+
Traps: STUN, STOV, BNDV

CMPB Compare bytes. This instruction scans two byte strings simultaneously until the compared bytes are unequal or until a specified number of comparisons have been made. CMPB expects a signed byte count in A, a DB or PB relative displacement for a source byte address in B, and a DB relative displacement for a target byte address in C. As long as the word count in A has not been counted to zero, the comparison proceeds as follows: The content of the byte address location specified by DB + B or PB + B is compared with the content of the byte address location specified by DB + C. If the byte count in A is positive, the source and target displacement values in B and C are incremented by one after each comparison, and the byte count is decremented by one. If the byte count in A is negative, the source and target displacement values in B and C are decremented by one after each comparison, and the byte count is incremented by one. Note that the byte count is always changed by one toward zero. The instruction terminates when either a comparison fails or the byte count in the TOS reaches zero. The Condition Code is set to a special pattern to indicate the terminating condition. On termination, the instruction deletes from the stack the number of words (0, 1, 2, or 3) specified by the SDEC field of the instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	0	1		0	0		

PB/DB (bit 11); SDEC (bits 14–15)

Instruction Commentary 9
Move opcode: 5
Indicators: CCE if byte count = 0
CCG if target byte > source byte (final)
CCL if target byte < source byte (final)

Addressing modes: Byte addressing
DB+ or PB+ for source
DB+ for target
Traps: STUN, STOV, BNDV, BNDV on P relative if user or privileged

SCW Scan while memory bytes equal test byte. The SCW instruction expects the TOS to contain a test character in the right byte and a terminal character in the left byte. The second word of the stack contains a DB relative displacement for a source byte address. The source byte is tested against the test character. If they are equal the source byte address is incremented and the next byte is tested. This continues until a source byte is found that is not the same as the test character. If the last character scanned is the same as the terminal character, the Carry bit is set; if not, the Carry bit is cleared. On completion of the scan, the instruction deletes from the stack the number of words (0, 1, 2, or 3) specified in the SDEC field of the instruction.
Instruction Commentary 9
Move opcode: 2, bits 11,12 = 10
Indicators: Carry
CCB on the last character scanned
Addressing mode: Byte addressing, DB+
Traps: STUN, STOV, BNDV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	0	1	0	1	0	0	SDEC	

SCU Scan until memory byte equals test byte or terminal byte. The SCU instruction expects the TOS to contain a test character in the right byte and a terminal character in the left byte. The second word of the stack contains a DB relative displacement for a source byte address. The source byte is tested against the test and terminal characters. If the source byte differs from both of these characters, the byte address is incremented and the next byte is tested. This continues until either the test character of the terminal character is encountered. The address of character remains in the second word of the stack. If the last character scanned was the same as the test character, the Carry bit is cleared; if it was the same as the terminal character, Carry is set. On completion of the scan, the instruction deletes from the stack the number of words (0, 1, 2, or 3) specified in the SDEC field of the instruction.
Instruction Commentary 9
Move opcode: 3, bits 11,12 = 10
Indicators: Carry
Addressing mode: byte addressing, DB+
Traps: STUN, STOV, BNDV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	0	1	1	1	0	0	SDEC	

MVBL Move words from DB+ to DL+. This instruction transfers a specified number of words from the DB+ area of the data segment to the DL+ area. The instruction expects a signed word count in A, a DB relative displacement for a source address in B, and a DL relative displacement for a target address in C. As long as the word count in A has not been counted to zero, the transferring of data will continue as follows: The contents of the memory location specified by DB + B is transferred to the location specified by DL + C. If the word count in A is positive, the source and target displacement values in B

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	0	1	0	0	0	0	SDEC	

and C are incremented by one on each transfer, and the word count is decremented by one. If the word count in A is negative, the source and target displacement values in B and C are decremented by one on each transfer, and the word count is incremented by one. Note that the word count is always changed by one toward zero. On completion of the block transfer, the instruction deletes from the stack the number of words (0, 1, 2, or 3) specified by the SDEC field of the instruction. This instruction can use split stack.

Instruction Commentary 10
Move opcode: 2, bits 11,12 = 00
Indicators: unaffected
Addressing modes: DB+ for source
DL+ for target
Traps: STUN, STOV, MODE
This is a privileged instruction.

MVLB Move words from DL+ to DB+. This instruction transfers a specified number of words from the DL+ area of the data segment to the DB+ area. The instruction expects a signed word count in A, a DL relative displacement for a source address in B, and a DB relative displacement for a target address in C. As long as the word count in A has not been counted to zero, the transferring of data will continue as follows: The contents of the memory location specified by DL + B is transferred to the location specified by DB + C. If the word count in A is positive, the source and target displacement values in B and C are incremented by one on each transfer, and the word count is decremented by one. If the word count in A is negative, the source and target displacement values in B and C are decremented by one on each transfer, and the word count is incremented by one. Note that the word count is always changed by one toward zero. On completion of the block transfer, the instruction deletes from the stack the number of words (0, 1, 2, or 3) specified by the SDEC field of the instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	0	1	1	0	0			

SDEC (bits 14–15)

Instruction Commentary 10
Move opcode: 3, bits 11,12 = 00
Indicators: unaffected
Addressing mode: DL+ for source
DB+ for target
Traps: STUN, STOV, MODE
This is a privileged instruction.

MABS Move using absolute addresses. This instruction expects to find a signed word count in A, an absolute source address in C and B (memory bank address in C), and an absolute target address in E and D (memory bank address in E). Words from the source area are moved into the target area with increasing addresses if the word count in A is positive or with decreasing addresses if the word count in A is negative. The positive word count in A is decremented towards zero or the negative word count is incremented towards zero with each word transferred. The transfer of words terminates when the word count reaches zero, then the instruction pops the number of words (0 through 7) specified in the SDEC field from the stack.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	0	1	0	0	1			

SDEC (bits 13–15)

S-4	E	Target bank address
S-3	D	Absolute target address
S-2	C	Source bank address
S-1	B	Absolute source address
S	A	Signed (+ or -) word count

Move opcode: 2, bits 11,12 = 01
Indicators: unaffected
Traps: MODE, STUN
This is a privileged instruction.

MTDS Move to data segment. This instruction expects to find a positive word count in A which represents the size of the block of words to be transferred, a DB-relative address in B to be used in calculating the source address of the first word to be transferred, and an offset into a target data segment in C to be used with the target data segment number in D to determine the address of the first target word location. The DST pointer is fetched from memory location 1 and added to four times the target data segment number in D to determine the desired target DST entry. A target address for storing the first word is then formed by adding the offset into the data segment contained in C to the segment base address contained in the fourth word of the DST entry. (The memory bank address for the segment is the third word of the DST entry.) A DB-relative source word address is formed by adding the DB address of the source segment and the DB-relative address in B. The source word pointed to is moved to the target location, the address pointers are incremented to point to the next source and data words, and the word count is decremented. Words from the source area continue to be moved to the target area until the word count reaches zero. The stack is then popped by the number of words (0 through 7) specified in the SDEC field.
Move opcode: 2, bits 11,12 = 11
Indicators: unaffected
Traps: MODE, DSTV, STUN, ABS DST
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	0	1	0	1	1			
													SDEC		

S-3	D	Target data segment number
S-2	C	Offset into target data segment
S-1	B	Source DB-relative address
S	A	Positive word count

MFDS Move from data segment. This instruction expects to find a positive word count in A which represents the size of the block of words to be transferred, an offset into a source data segment in B to be used with the source data segment number in C to determine the address of the first source data word, and a DB-relative address in D to be used in calculating the target address for the first word transfer. The DST pointer is fetched from memory location 1 and added to four times the number in C to determine the desired DST entry. A source word is then formed by adding the offset into a data segment value contained in B to the segment base address contained in the fourth word of the DST entry. (The memory blank address for the segment is the third word of the DST entry.) A DB-relative target word address is formed by adding the target DB address and the DB-relative address in D. The target word address thus pointed to receives a word from the source word address, the address pointers are incremented to point to the next source word and target word locations, and the word count is decremented. Words from the source area continue to be moved to the target area until the word count reaches zero. The stack is then popped by the number of words (0 through 7) specified in the SDEC field.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	0	1	1	1	1			
													SDEC		

S-3	D	Target DB-relative address
S-2	C	Source data segment number
S-1	B	Offset into source data segment
S	A	Positive word count

Move opcode: 3, bits 11,12 = 11
Indicators: unaffected
Traps: MODE, DSTV, STUN, ABS DST
This is a privileged instruction.

MDS Move using data segments. This instruction expects to find a signed word count in A which represents the size of the block of words to be transferred, an offset into a source data segment in B to be used with the source data segment number in C to determine the address of the first source data word, and an offset into a target data segment in D to be used with the target data segment number in E to determine the address of the first target word. The DST pointer is fetched from memory location 1 and added to four times the source data segment number in C to point to the desired source DST entry and to four times the target data segment number in E to point to the desired target DST entry. A source word address is then formed by adding the offset contained in B to the segment base address contained in the fourth word of the source DST entry and a target word address is formed by adding the offset contained in D to the segment base address contained in the fourth word of the target DST entry. (The memory bank address for a data segment is the third word of the DST entry.) Words from the source area are moved into the target area with increasing addresses if the count in A is positive or with decreasing addresses if the count in A is negative. The positive count is decremented or the negative count is incremented with each word transferred. The transfer of words terminates when the count in A reaches zero, then the instruction pops the number of words (0 through 7) specified in the SDEC field from the stack.
Move opcode: 3, bits 11,12 = 01
Indicators: unaffected
Traps: MODE, DSTV, STUN, ABS DST
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	0	1	1	0	1	SDEC		

S-4	E	Target data segment number
S-3	D	Offset into target data segment
S-2	C	Source data segment number
S-1	B	Offset into source data segment
S	A	Signed (+ or -) word count

PRIVILEGED MEMORY REFERENCE INSTRUCTIONS

PLDA Privileged load from absolute address. The content of the Index register is a 16-bit absolute address in bank 0; the content of this address is pushed onto the stack.
Mini-opcode: 15, bit 15 = 0
Indicators: CCA
Addressing mode: absolute
Traps: STOV, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	0	1	0	0	0	0

PSTA Privileged store into absolute address. The content of the Index register is a 16-bit absolute address in bank 0; the top word of the stack is stored into memory at the address, and then deleted from the stack.
Mini-opcode: 15, bit 15 = 1
Indicators: unaffected
Addressing mode: absolute
Traps: STUN, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	0	1	0	0	0	1

LST Load from system table. The X register contains a value which is used to index into a table pointed to by the contents of location %1000+K if K is non-zero, or by the contents of location %1000+A if K is zero. The table pointer itself is also relative to location %1000. The data accessed in the table is pushed onto the stack if K is non-zero or replaces A if K is zero.
Special opcode: 00
Indicators: CCA
Traps: STUN, STOV, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	0	0	0	K			

SST Store into system table. The X register contains a value which is used to index into a table pointed to by the contents of location %1000+K if K is non-zero, or by the contents of location %1000+A if K is zero. The table pointer itself is also relative to location %1000. The data contained in A if K is non-zero or in B if K is zero is stored into the calculated address. The stack is then popped by one if K is non-zero or by two if K is zero.
Special opcode: 15
Indicators: unaffected
Traps: STUN, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	1	0	1	K			

LSEA Load single word from extended address. A bank address is in B and A is a 16-bit absolute address of a location in that bank. The word at that address is pushed onto the stack.
Mini-opcode: 16, bits 14,15 = 00
Indicators: CCA
Addressing mode: absolute
Traps: STUN, STOV, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0

SSEA Store single word into extended address. A bank address is in C and B is a 16-bit absolute address of a location in that bank. The TOS is stored in the location pointed to and the stack is popped.
Mini-opcode: 16, bits 14,15 = 01
Indicators: unaffected
Addressing mode: absolute
Traps: STUN, STOV, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	1

LDEA Load double word from extended address. A bank address is in B and A is a 16-bit absolute address of a location in that bank. The double word at that address is pushed onto the stack. The word in B is the most significant.
Mini-opcode: 16, bits 14,15 = 10
Indicators: CCA
Addressing mode: absolute
Traps: STUN, STOV, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	1	0	0	0	1	0

SDEA Store double word into extended address. A bank address is in D and C is a 16-bit absolute address of a location in that bank. The double word on the top of the stack is stored in the location pointed to and popped from the stack. The word in B is the most significant.
Mini-opcode: 16, bits 14,15 = 11
Indicators: unaffected
Traps: STUN, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	1	0	0	0	1	1

IMMEDIATE INSTRUCTIONS

LDI Load immediate. The immediate operand N is pushed onto the stack. The value of N is given in the argument field of the instruction, and is expressed as a positive integer in the range 0 through 255.
Sub-opcode 2: 02
Indicators: CCA on the new TOS
Traps: STOV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	1	0								

Immediate Operand

LDXI Load X immediate. The Index register is loaded with the immediate operand N. The value of N is given in the argument field of the instruction, and is expressed as a positive integer in the range 0 through 255.
Sub-opcode 2: 03
Indicators: unaffected
Traps: None

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	1	1								

Immediate Operand

CMPI Compare immediate. The Condition Code is set to pattern C as a result of the comparison of the TOS with the immediate operand N. The value of N is given in the argument field of the instruction, and is expressed as a positive integer in the range 0 through 255. The TOS is deleted.
Sub-opcode 2: 04
Indicators: CCC
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	1	0	0								

Immediate Operand

ADDI Add immediate. The immediate operand N is added to the TOS in integer form, and the sum replaces the TOS. The value of N is given in the argument field of the instruction, and is expressed as a positive integer in the range 0 through 255.
Sub-opcode 2: 05
Indicators: CCA on the new TOS, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	1	0	1								

Immediate Operand

SUBI Subtract immediate. The immediate operand N is subtracted from the TOS in integer form, and the result replaces the TOS. The value of N is given in the argument field of the instruction, and is expressed as a positive integer in the range 0 through 255.
Sub-opcode 2: 06
Indicators: CCA on the new TOS, Carry, Overflow
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	1	1	0								

Immediate Operand

MPYI Multiply immediate. The immediate operand N is multiplied with the TOS in integer form; the 16-bit integer result replaces the TOS. The value of N is expressed as a positive integer in the range 0 through 255.
Sub-opcode 2: 07
Indicators: CCA on the new TOS, Overflow
Traps: STUN, STOV, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	1	1	1								

Immediate Operand

DIVI Divide immediate. The immediate operand N is divided into the TOS in integer form; the 16-bit integer quotient replaces the TOS. The value of N is expressed as a positive integer in the range 0 through 255.
Sub-opcode 2: 10
Indicators: CCA on the new TOS
Traps: STUN, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	1	0	0	0								

Immediate Operand

LDNI Load negative immediate. The immediate operand N is two's complemented and pushed onto the stack as a negative integer. The value of N is expressed as a positive integer in the range 0 through 255.
Sub-opcode 2: 12
Indicators: CCA on the new TOS
Traps: STOV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	1	0	1	0	Immediate Operand							

LDXN Load X negative immediate. The Index register is loaded with the 16-bit two's complement of the immediate operand N. The value of N is expressed as a positive integer in the range 0 through 255.
Sub-opcode 2: 13
Indicators: unaffected
Traps: None

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	1	0	1	1	Immediate Operand							

CMPN Compare negative immediate. The Condition Code is set to pattern C as a result of the comparison of the TOS with the two's complement of the immediate operand N. The value of N is expressed as a positive integer in the range 0 through 255. The TOS is deleted.
Sub-opcode 2: 14
Indicators: CCC
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	1	1	0	0	Immediate Operand							

ADXI Add immediate to X. The immediate operand N is added to the content of the Index register in integer form. The sum replaces the Index register content. The value of N is expressed as a positive integer in the range 0 through 255.
Sub-opcode 3: 05
Indicators: CCA on X
Traps: None

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	1	0	1	Immediate Operand							

SBXI Subtract immediate from X. The immediate operand N is subtracted from the content of the Index register in integer form. The result replaces the Index register content. The value of N is expressed as a positive integer in the range 0 through 255.
Sub-opcode 3: 06
Indicators: CCA on X
Traps: None

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	1	1	0	Immediate Operand							

ORI Logical OR immediate. The immediate operand N is expanded to 16 bits with high order zeros and merged (inclusive OR) with the TOS; the result replaces the TOS. The value of N is expressed as a positive integer in the range 0 through 255.
Sub-opcode 3: 15
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	1	1	0	1	Immediate Operand							

XORI Logical exclusive OR immediate. The immediate operand N is expanded to 16 bits with high order zeros and is combined by exclusive OR with the TOS; the result replaces the TOS. The value of N is expressed as a positive integer in the range 0 through 255.
Sub-opcode 3: 16
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	1	1	1	0	Immediate Operand							

ANDI Logical AND immediate. The immediate operand N is expanded to 16 bits with high order zeros and is combined by logical AND with the TOS; the result replaces the TOS. The value of N is expressed as a positive integer in the range 0 through 255.
Sub-opcode 3: 17
Indicators: CCA
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	1	1	1	1								
								Immediate Operand							

REGISTER CONTROL INSTRUCTIONS

SETR Set registers. The registers specified by bits 8 through 15 of the instruction are filled by an absolute value from the TOS for the Index, Status, DB, DB-Bank, and S-Bank registers, and an absolute value computed by adding (new) DB to the TOS (displacement value) for the others. If more than one register (or displacement) is specified, the registers will be loaded in the order shown below, such that if all nine were specified, the S-Bank register would receive the first TOS and the value for S would be computed from the ninth TOS. The TOS is deleted after each register is set. SETR is a privileged instruction except for setting of the Index register, Q, S, and bits 2 and 4 through 7 of the Status register. (The Status bits are user traps enable/disable, Overflow, Carry, and Condition Code. Attempts to set other bits of Status will be ignored and will not cause a MODE trap.)

*If bit 9	=	1, load S-Bank from TOS
*If bit 9	=	1, load DB from TOS
*If bit 10	=	1, load DL from (DB+TOS)
*If bit 11	=	1, load Z from (DB+TOS)
*If bit 12	=	1, load Status from TOS
If bit 12	=	1, and not privileged mode: load Status bits 2 and 4 thru 7 from same bits of TOS
If bit 13	=	1, load Index register from TOS
If bit 14	=	1, load Q from (DB+TOS)
If bit 15	=	1, load S from (DB+TOS)

Sub-opcode 2: 17
Indicators: unaffected (may be changed if bit 12 = 1)
Traps: STUN, STOV, MODE
*These are privileged operations.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	1	1	1	1								
								S-Bank	DB	DL	Z	Sta	X	Q	S

PSHR Push registers. The content of a register (or the displacement it represents) specified by any bit 8 through 15 is pushed onto the stack. If more than one register (or displacement) is specified, the contents will be stacked in the order shown below, such that if all nine were specified, S-Bank would be on the TOS after execution, DB next, etc. Note that when S-DB is pushed, the value stacked will be as it existed before the execution of this instruction. Stack overflow occurs if the original S+9 exceeds Z, regardless of the number of registers pushed.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	1	0	0	1								
								S-Bank	DB	DL	Z	Sta	X	Q	S

If bit 15 = 1, push S-DB
If bit 14 = 1, push Q-DB
If bit 13 = 1, push Index register
If bit 12 = 1, push Status register
If bit 11 = 1, push Z-DB
If bit 10 = 1, push DL-DB
*If bit 9 = 1, push DB Bank and DB register
*If bit 8 = 1, push S-Bank
Sub-opcode 2: 11
Indicators: unaffected
Traps: STOV, MODE
*These are privileged operations.

XCHD Exchange DB and TOS. This instruction expects a new DB value on the TOS and a new DB-Bank at TOS-1. The current DB-Bank, DB replaces these values in TOS-1, TOS, while the new values are placed in DB-Bank, DB.
Special opcode: 03, bits 12-15 = 0000
Indicators: unaffected
Traps: STUN, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0

ADDS Add to S. The immediate operand N is added to S unless N is zero; if N is zero, the TOS content, minus one, is added to S instead.
Instruction Commentary 11
Sub-opcode 3: 12
Indicators: unaffected
Traps: STUN, STOV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	1	0	1	0	Immediate Operand							

SUBS Subtract from S. The immediate operand N is subtracted from S unless N is zero; if N is zero, the TOS content, plus one, is subtracted from S instead.
Instruction Commentary 11
Sub-opcode 3: 13
Indicators: unaffected
Traps: STUN, STOV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	1	0	1	1	Immediate Operand							

RCLK Read clock. This instruction pushes the contents of the Process Clock register onto the top of the stack.
Special opcode: 5, bits 12-15 = 0001
Indicators: unaffected
Traps: STOV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	1	0	1	0	0	0	1

SCLK Store clock. This instruction expects to find a 16-bit word on the top of the stack that it uses to set the Process Clock register. The stack is then popped.
Special opcode: 4, bits 12-15 = 0001
Indicators: unaffected
Traps: STUN, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	1

PROGRAM CONTROL AND SPECIAL INSTRUCTIONS

SCAL Subroutine call. Control is transferred to the location pointed to by the evaluation of the local label at PL– N, unless N is zero; if N is zero the local label is taken from the TOS and then deleted. The return address is then pushed onto the stack. Only local labels are allowed; non-local label gives STT Violation trap.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	1	N							

Instruction Commentary 12
Sub-opcode 3: 01
Indicators: unaffected
Addressing modes:
Indirect via: PL – N (if N ≠ 0)
TOS (if N = 0)
Local Label: PB+
Traps: STUN, STOV, STTV, BNDV if user or privileged

PCAL Procedure call. Control is transferred to the location pointed to by the evaluation of the program label at PL – N, unless N is zero, the program label is taken from the TOS and then deleted. Then a four word stack marker is placed on the stack, and Q and S are updated to point at this new marker. The program label may be local or external. If the Trace bit is on in the target CST entry, a call will be made to Trace, segment #1, STT #32 (decimal). If a privileged user is calling a user segment, it will run in privileged mode.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	1	0	N							

Instruction Commentary 13
Sub-opcode 3: 02
Indicators: unaffected
Addressing modes:
Indirect via: PL – N (if N ≠ 0)
TOS (if N = 0)
Local Label: PB+
External Label: via CST to local label in target segment
Traps: STUN, STOV, CSTV, STTV, ABS CST, TRACE, UNCALL, BNDV if user or privileged

SXIT Exit from subroutine. This instruction is used to return from a subroutine called by the SCAL instruction. The SXIT instruction assumes that the return address is on the TOS, and returns program control to this address. The TOS is then deleted, plus N number of subroutine parameters. The value of N may be any number from 0 through 255.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	1	0	0	N							

Instruction Commentary 12
Sub-opcode 3: 04
Indicators: unaffected
Traps: STUN, STOV, BNDV if user or privileged

EXIT Exit from procedure. This instruction is used to return from a procedure called by the PCAL instruction or by some interrupts. A normal exit occurs by restoring the return address to P, restoring the previous contents of the Index and Status registers, and deleting all stack variables incurred by the called routine plus its marker, plus N number of procedure parameters. The value of N may be any number from 0 to 255 for exits from PCAL routines; it must be 0 for exits from interrupt routines. If bit 0 of the return-P marker word is a "1", control is transferred to Trace, segment #1, STT #32 (decimal).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	1	1	N							

Instruction Commentary 13
Sub-opcode 3: 03
Indicators: Restored to values before PCAL
Traps: STUN (going to user mode), STOV, MODE, CSTV, TRACE, ABS CST, BNDV if user or privileged

LLBL Load label. The label in the Segment Transfer Table (STT) at PL– N is loaded onto the TOS. The value N is a displacement given in the argument field of the instruction. If the label is local, it is converted to external type when loaded. To be valid, the value N must point to a location which is actually in the STT (i.e., N ≤ STTL) in all cases; additionally, in the case of local labels, N must not exceed octal 177 (decimal 127), since this is the maximum range for the STT # in the external label result.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	1	1	1								

$\underbrace{\text{bits 8–15}}_{\text{Displacement PL-}}$

Instruction Commentary 14
Sub-opcode 3: 07
Indicators: unaffected
Addressing mode: PL–
Traps: STOV, STTV

IXIT Interrupt exit. This instruction is used to exit from those interrupt service routines which always run on the Interrupt Control Stack (ICS). This results in a return to the interrupted process (which may be another interrupt or the Dispatcher) or a transfer to the Dispatcher's entry point. The action taken depends in part on the sequence of DISP, PSDB, and PSEB instructions which have been executed. IXIT is also used by the Dispatcher to exit to a process being launched.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	0

Instruction Commentary 15
Mini-opcode: 17, bits 12-15 = 0000
Indicators: Restored to those before interrupt or as specified for the Dispatcher
Traps: MODE, STOV, CSTV, TRACE, ABS CST, BNDV if user or privileged
This is a privileged instruction.

DISP Dispatch. This instruction is used to transfer to the Dispatcher's entry point; or to request such a transfer if executed while on the ICS or within the range of a PSDB-PSEB pair.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	0

Instruction Commentary 15
Special opcode: 03, bits 12-15 = 0010
Indicators: See instruction commentary.
Traps: MODE, CSTV, TRACE, ABS CST, BNDV if user or privileged
This is a privileged instruction.

PSDB Pseudo interrupt disable. The PSDB and PSEB instructions are used in pairs and may be nested. They are used to prevent a dispatch during critical sections of code, and to avoid unnecessary restarting of the Dispatcher. The effect of any DISP instructions executed within the range of a PSDB-PSEB pair located outside of the Dispatcher is postponed until the numbers of PSDB and PSEB instructions executed are equal. DISP is effectively a NOP when executed within the range of a PSDB-PSEB pair located in the Dispatcher.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	1

Instruction Commentary 15
Special opcode: 03, bits 12-15 = 0001
Indicators: unaffected
Traps: MODE
This is a privileged instruction.

PSEB Pseudo interrupt enable. See description of PSDB instruction just given.
Instruction Commentary 15
Special opcode: 03, bits 12-15 = 0011
Indicators: See instruction commentary.
Traps: MODE, CSTV, TRACE, ABS CST, BNDV if user or privileged
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	0	1	1	0	0	1	1

PAUS Pause. The computer hardware pauses; interrupts may occur. Bits 12 through 15 are ignored.
Special opcode: 01
Indicators: unaffected
Traps: MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0

Not Used

HALT The computer hardware halts; interrupts may not occur and manual intervention is required to restart the computer. Bits 12 through 15 are ignored.
Special opcode: 17
Indicators: unaffected
Traps: MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	1	1	1				

K

LOCK* This instruction provides a means for one CPU to lock out another CPU in a two-CPU system. A typical application would be in a multiprogramming system when a CPU is going to use a critical portion of code that is shared by both CPUs. The LOCK instruction tests the lockword pointed to by the contents of the Index register and at the same time sets a bit in the lockword corresponding to its CPU number. Bit 15 is set for CPU number one and bit 14 is set for CPU number two. If the lockword contents was zero, no one had the resource, and the CPU executing the LOCK instruction gets the resource. If the lockword was not equal to zero, indicating that the other CPU has the resource, the instruction goes into a pause mode and will require an interrupt to restart the LOCK instruction from the beginning.
Mini-opcode: 17, bits 12-15 = 0010
Indicators: unaffected
Traps: MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	1	1	0	0	0	1

UNLK* This instruction releases a resource previously locked by the same CPU which executed the LOCK instruction to secure the resource. The lockword is fetched and an interrupt is sent to the other CPU if it is in the pause mode after an unsuccessful attempt to execute a LOCK instruction. The interrupt thus "awakens" the other resource requester and the instruction clears the lockword releasing the resource. The other CPU will then (successfully) re-execute its LOCK instruction.
Mini-opcode: 17, bits 12-15 = 0011
Indicators: unaffected
Traps: MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	1	1	0	0	1	1

*Series II Computer Systems only.

LLSH A bank address is in B and A contains a 16-bit absolute address in the bank which points into a linked list. Each double word link in the list is an absolute memory address which points to the next link. C contains a test word and D contains an offset which indicates the position, relative to each link, of a target number. At each step, the test word is compared to the target number. If the test word is logically less than or equal to the target number, the instruction terminates. Otherwise, the contents of B and A is replaced by the next link, the count in the Index register is decremented, and the instruction repeats.

Instruction Commentary 16
Mini-opcode: 14, bit 15 = 1
Indicators: CCL if terminated by X = 0
CCE if terminated by target ≥ C
CCG if terminated by target = $2^{16} - 1$
Addressing mode: absolute ± offset
Traps: STUN, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	1

XEQ Execute stack word. The content of the word in the stack at S-K is placed in the Current Instruction Register to be executed. After execution, control is returned to the instruction after the XEQ unless a transfer of control was executed (branch, PCAL, etc.). If the word to be executed is a Stack Op, only the first position (bits 4 through 9) may be used; bits 10 through 15 must be a NOP. The value of K is to 0 through 15 (decimal).

Instruction Commentary 17
Special opcode: 06
Indicators: set by the execution instruction
Traps: BNDV and traps possible during the executed instruction's execution

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	1	1	0	K			

RSW Read Switch register. The content of the Switch register is pushed onto the stack.

Mini-opcode: 14, bit 15 = 0
Indicators: CCA
Traps: STUN, STOV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0

PCN Push CPU number. This instruction pushes a number onto the stack identifying the type of CPU executing the PCN instruction. This will be either 1 or 2. (1 = Series II; 2 = Series III.)

Mini-opcode: 17, bits 12-15 = 0010
Indicators: unaffected
Traps: STOV, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	1	1	0	0	1	0

RCCR* Read system clock counter. The contents of the 12-bit system clock counter are pushed onto the stack.

Opcode: 00
Indicators: unaffected
Traps: Stack overflow

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WORD 1	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0
WORD 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

*Series 30/33 Computer Systems only.

SCLR* Set system clock limit. The lower 12 bits of the word on the top of the stack are loaded into the system clock limit register and the stack is popped.
Opcode: 01
Indicators: unaffected
Traps: none

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WORD 1	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0
WORD 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

TOFF* Hardware timer off. Turns the CPU hardware timer off. This timer is used to simulate both System and Process clocks, so turning it OFF will disable them. The timer will be turned OFF by the CPU on LOAD, RESTART, PON and PWF.
Opcode: 03
Indicators: unaffected
Traps: none

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WORD 1	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0
WORD 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

SINC* Set system clock interrupt. The most significant bit of the System Clock status register is set. If external interrupts are enabled, there is an immediate trap to seq. 1, STT 12. If external interrupts are disabled, then the system clock status register is incremented and no trap taken.
Opcode: 10
Indicators: unaffected
Traps: see above

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WORD 1	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0
WORD 2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

TON* Hardware timer on. Turns the CPU hardware timer on. (See TOFF.)
Opcode: 02
Indicators: unaffected
Traps: none

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WORD 1	0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0
WORD 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

*Series 30/33 Computer Systems only.

I/O INSTRUCTIONS

SED Set "enable/disable external interrupts" bit. The interrupt system is enabled or disabled according to the least significant bit (bit 15) of the instruction. If K is equal to 1, bit 1 of the Status register is set, thus enabling external interrupts. If K is equal to 0, bit 1 of the Status register is cleared, thus disabling external interrupts. If the instruction changes bit 1 of the Status register from 1 to 0, any pending interrupts will occur immediately following the SED instruction.
Special opcode: 02
Traps: MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	
															D/E
														K	

SMSK Set mask. The SMSK instruction assumes that the TOS contains the mask word and transmits this word to all device controllers. Each "1" bit in the mask word sets each Mask flip-flop in the group of device controllers which are specifically wired to be controlled by that bit. Each "0" bit in the mask clears each Mask flip-flop in its group. If there is an I/O error (no acknowledgement), the SMSK instruction sets CCL Condition Code, and leaves the mask on the TOS. If there is no I/O error, the SMSK instruction deletes the mask from the stack and sets the CCE Condition Code. The mask word is also stored in memory at location 7 for CPU #1 or at location %13 for CPU #2.
Special opcode: 04, bits 12-15 = 0000
Indicators: CCE if no error
CCL if error
Traps: STUN, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	1	0	0	0	0	0	0

RMSK Read mask. This instruction loads the 16-bit mask word from memory into the TOS.
Special opcode: 05, bits 12-15 = 0000
Indicators: unaffected
Traps: STOV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	1	0	1	0	0	0	0

SIO* Start I/O. The SIO instruction expects the absolute starting address of an I/O program to be on the TOS, and a device number to be in the stack at S-K. The instruction first checks if the device is ready for an SIO by checking bit 0 of the device controller's Status register. Bit 0 is the "SIO OK" bit. If it is ready (bit = "1"), the TOS is stored into the first location of the DRT entry for the device specified at S-K; an SIO command is then issued to the device controller to begin execution of its I/O program. If the device is not ready (bit 0 of the device status = "0"), the content of the device controller's Status register is pushed onto the stack and the Condition Code is set to CCG. If the device controller does not respond, the Condition Code is set to CCL and the instruction is terminated. If the device is ready, the TOS is deleted and the Condition Code is set to CCE.
Instruction Commentary 18
Special opcode: 07

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	0	1	1	1				
												K			

*Series II/III Computer Systems only.

Indicators: CCL = non-responding device controller
CCE = device ready
CCG = device not ready
Traps: STUN, STOV, MODE
This is a privileged instruction.

RIO* Read I/O. This instruction expects a device number to be given in the stack at S-K. RIO first checks if the device is ready by checking bit 1 of the device controller's Status register. If it is ready (bit = "1"), the 16-bit data word from the device is pushed onto the stack and the Condition Code is set to CCE. If it is not ready (bit = "0"), the content of the device controller's Status register is pushed onto the stack and the Condition Code is set to CCG. If the device controller does not respond to the readiness test, the Condition Code is set to CCL and the instruction is terminated.
Special opcode: 10
Indicators: CCL = non-responding device controller
CCE = device ready
CCG = device not ready
Traps: STOV, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	0	0	0	K			

WIO* Write I/O. This instruction assumes that the TOS contains a data word and expects a device number to be given in the stack at S-K. WIO first checks if the device is ready by checking bit 1 of the device controller's Status register. If it is ready (bit = "1"), the word is transmitted to the specified device and then deleted from the stack; the Condition Code is set to CCE. If it is not ready (bit = "0"), the content of the device controller's Status register is pushed onto the stack and the Condition Code is set to CCG. If the device controller does not respond, the Condition Code is set to CCL and the instruction is terminated.
Special opcode: 11
Indicators: CCL = non-responding device controller
CCE = device ready
CCG = device not ready
Traps: STUN, STOV, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	0	0	1	K			

TIO* Test I/O. This instruction expects a device number to be given in the stack at S-K. TIO obtains a copy of the device status word from the device controller, pushes it onto the stack, and sets the Condition Code to CCE. If the device controller does not respond, the Condition Code is set to CCL and the instruction is terminated.
Special opcode: 12
Indicators: CCE = responding device controller
CCL = non-responding device controller.
Traps: STOV, MODE
This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	0	1	0	K			

CIO* Control I/O. This instruction assumes that the TOS contains a control word and expects a device number to be given in the stack at S-K. CIO transmits the TOS to the specified device controller, along with a CIO signal. If the device controller acknowledges receiving the word, the TOS is deleted and the Condition Code is set to CCE.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	0	1	1	K			

*Series II/III Computer Systems only.

If the device controller does not respond, the Condition Code is set to CCL and the instruction is terminated.

Special opcode: 13

Indicators: CCE = responding device controller
CCL = non-responding device controller

Traps: STUN, MODE

This is a privileged instruction.

CMD* Command. This instruction assumes that the TOS contains a 16-bit data word to be sent to a system hardware module and expects a command word in the stack at S-K. Bits 13 through 15 of the command word specify the module number, and bits 10 and 11 are used to specify a module command. (The four possible commands are interpreted by the target module and do not form a part of this instruction's definition.) CMD sends the 16-bit data word and the 2-bit command over the central data bus to the specified module, and then deletes the TOS. (Note: if the destination module is not ready, the CPU will not proceed until that module becomes ready.)

Special opcode: 14

Indicators: unaffected

Traps: STUN, MODE

This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	1	0	0	K			

SIN* Set interrupt. This instruction expects a device number to be given in the stack at S-K. SIN sets the Interrupt Request flip-flop in the specified device controller and sets the Condition Code to CCE. If the device controller does not respond, the Condition Code is set to CCL and the instruction is terminated.

Special opcode: 16

Indicators: CCE = responding device controller
CCL = non-responding device controller

Traps: MODE

This is a privileged instruction.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	0	0	0	0	1	1	1	0	K			

DUMP** Load Soft Dump program. A dumpload from the device in (S) is initiated, following the LOAD/START/DUMP procedure. (See Instruction Commentary 20.)

The device is assumed to be a disk; 1 sector (#3 for DUMP) is loaded from device (S), head # (S-1), and executed as a channel program. The effect is the same as using the "DUMP" front panel keys. If the instruction is successful, the result is a LOAD trap to SEG 1, STT 24; any error results in a system HALT (See Instruction Commentary 21.)

Opcode: 12

Traps: LOAD; Stack Underflow; Non-responding device

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WORD 1	0	0	1	0	0	0	0	0	1	1	0	0	0	0	1	0
WORD 2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

WIOC** Write I/O channel. This instruction expects an IMB "read channel" command in S-1 and a data word in (S). If the abort bit is not set for the device the data word in TOS and the command in (S-1) are sent to the channel, or channels if global, and the data word and command are popped from the stack. This instruction provides full software control of the channel and devices of any type.

Opcode: 03

Indicators: If error then CCL else CCE

Traps: Stack Underflow; Non-responding device.

This is a privileged instruction

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WORD 1	0	0	1	0	0	0	0	0	1	1	0	0	0	0	1	0
WORD 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

*Series II/III Computer Systems only.
**Series 30/33 Computer Systems only.

SIOP** Start I/O program. This instruction expects a channel program pointer in (S) and channel/device number in (S-1). The third word of the device DRT entry (DRT3) is read with a semaphore read. This delays execution of the instruction by a possible independent program channel until all the information is in place. If bit 2 of DRT3 (the abort bit) is 1, the instruction is aborted and CCL is set. If the channel program is halted (if bits 0,1 of DRT3 are both 0), or if an HIOP instruction has been issued but not yet serviced and the channel is in a wait instruction state (bits 0,1 of DRT3 are 0 and 1 and bit 15 of DRT3 is a 1), then the channel program pointer in (S) is placed in DRT0 of that device, bits 0,1 of DRT3 are set to 1,1 (SIO starting state), an SIO command is sent to the channel and CCE is set. Otherwise if the above conditions are not met then CCG is set.

Opcode: 00
Indicators: Condition Code
Traps: Stack Underflow; Non-responding device

This is a privileged instruction.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WORD 1	0	0	1	0	0	0	0	0	1	1	0	0	0	0	1	0
WORD 2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

INIT** Initialize I/O channel. The INIT instruction initializes the channel designated by bits 9-12 in the TOS by

Terminating operations in progress on the channel;
Clearing the channels interrupt enable bit;
Setting channel registers to defined initial values;
Setting the channel HP-IB bus to the idle state;
Clearing the 4th word of every DRT entry for this channel;
Clearing the mask bit for that channel in mem loc. 7.

Devices controlled by I/O software can be cleared only by being issued a DCL or SDC Interface Command (refer to HP Interface Bus Standards).

Opcode: 06
Indicators: If not system controller then CCG, else CCE
Traps: Stack Underflow; Non-responding device
This is a privileged instruction

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WORD 1	0	0	1	0	0	0	0	0	1	1	0	0	0	0	1	0
WORD 2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0

MCS** Read memory controller. An IMB "MCRS" operation is done. Address lines are set from (S-1), (S). If address bit 13 is 0, the returned data word is pushed on the stack; otherwise the data word is put in TOSA and (S) is incremented but is actually not written to memory. (Note: this means that if the returned word is to be saved or used, it must be recovered from TOSA using a "STAX" instruction (or something similar)). The actual functions performed by the MCRS instruction are dependent upon the particular memory controller used in the system.

Opcode: 07
Traps: Stack Underflow; Non-responding device

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
WORD 1	0	0	1	0	0	0	0	0	1	1	0	0	0	0	1	0
WORD 2	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1

**Series 30/33 Computer Systems only.

HIOP** Halt I/O program. This instruction expects a device number on the top of the stack. If DRT3 (0:2) of that device indicates that the device channel program is starting or running, a halt I/O program command is sent to the channel to stop execution of that device's channel command program at the occurrence of the next WAIT channel command. If starting, a "halting in WAIT" state is set to properly terminate the channel program. If now the channel program is halting but not yet halted and not in a WAIT instruction, CC is set to CCG. All other states are set to CCE including the already halted state. When halted, the DRT entry for that device points at the WAIT instruction at which the channel program halted, bits 0 and 1 of DRT3 are set to 0. If the channel program was not at a wait instruction when the HIOP was issued, an interrupt request will be generated when the channel program is halted.

Opcode: 01

Indicators: Condition Code

Traps: Stack Underflow; Non responding device

This is a privileged instruction

WORD 1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

WORD 2

SEML** Semaphore load. The contents of the memory location addressed by (S-1), (S) are read by a special memory operation that reads the location and replaces its contents with 1's in one step. The original contents of the location is pushed onto the stack.

Opcode: 10

Indicators: CCA on new TOS.

Carry set if (E) = 1

Traps: Stack Underflow; Non-responding device

WORD 1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

WORD 2

RIOC** Read I/O channel. This instruction expects an IMB "read channel" command to be on the top of the stack. If the abort bit is not set for the device, the RIOC command from the TOS is sent to the channel (or channels if global command) and the data read is pushed onto the stack

Opcode: 02

Indicators: Condition Code

Traps: Stack Underflow; Non-responding device

This is a privileged instruction.

WORD 1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

WORD 2

STRT** Initiate warmstart. A warmstart from the device in (S) is initiated, following the LOAD/START/DUMP procedure. (See Instruction Commentary 20.) The device is assumed to be a disk; 1 sector (#2 for STRT) is loaded from device (S), head # (S-1), and executed as a channel program. The effect is the same as using the "START" front panel key. If the instruction is successful, the result is a LOAD trap to SEG1, STT24; any error results in a SYSTEM HALT (See Instruction Commentary 21.)

Opcode: 11

Traps: Stack Underflow; Non-responding device

WORD 1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	0	1	1	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1

WORD 2

**Series 30/33 Computer Systems only.

LOOP CONTROL INSTRUCTIONS

TBA Test and branch, limit in A. This instruction expects the top three elements of the stack to be initialized as follows: A contains a limit, B contains a step size, and C contains a DB+ relative displacement for the address of a variable. TBA tests the variable against the limit. If the limit is not exceeded, control is transferred to the branch address at P ± displacement. If the limit is exceeded, the top three elements of the stack are deleted and execution continues at P + 1.

Instruction Commentary 19
Memory opcode: 05, bits 4,5,6 = 000
Indicators: unaffected
Addressing mode: P relative (+/–)
Traps: STUN, STOV, BNDV, BNDV on P if user or privileged

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	1	0	1	0	±								

Displacement (bits 8–15)

MTBA Modify variable, test and branch, limit in A. This instruction expects the top three elements of the stack to be initialized as follows: A contains a limit, B contains a modifying step size, and C contains a DB+ relative displacement for the address of a variable. MTBA adds the step size to the variable in integer form, replaces the old variable with this new sum, and tests the new sum against the limit. If the limit is not exceeded, control is transferred to the branch address at P ± displacement. If the limit is exceeded, the top three elements of the stack are deleted and execution continues at P + 1.

Instruction Commentary 19
Memory opcode: 05, bits 4,5,6 = 010
Indicators: unaffected
Addressing mode: P relative (+/–)
Traps: STUN, STOV, BNDV, BNDV on P if user or privileged.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	1	1	0	0	±								

Displacement

TBX Test and branch, variable in X. This instruction requires that the Index register contains the variable and that the top two elements of the stack are initialized as follows: A contains a limit and B contains a step size. TBX tests the variable in X against the limit. If the limit is not exceeded, control is transferred to the branch address at P ± displacement. If the limit is exceeded, the top two elements of the stack are deleted and execution continues at P + 1.

Instruction Commentary 19
Memory opcode: 05. bits 4,5,6 = 100
Indicators: unaffected
Addressing mode: P relative (+/–)
Traps: STUN, BNDV if user or privileged

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	1	0	0	0	±								

Displacement

MTBX Modify variable in X, test and branch. This instruction requires that the Index register contains the variable and that the top two elements of the stack are initialized as follows: A contains a limit and B contains a modifying step size. MTBX adds the step size to the variable in integer form, replaces the old Index register contents with this new sum, and tests the new sum against the limit. If the limit is not exceeded, control is transferred to the branch address at P ± displacement. If the limit is exceeded, the top two elements of the stack are deleted and execution continues at P + 1.

Instruction Commentary 19
Memory opcode: 05, bits 4,5,6 = 110
Indicators: unaffected
Addressing mode: P relative (+/–)
Traps: STUN, BNDV if user or privileged

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	1	1	1	0	±								

Displacement

MEMORY ADDRESS INSTRUCTIONS

LOAD **Load word onto stack. The content of the effective address location is pushed onto the stack.**
Memory opcode: 04
Indicators: CCA
Addressing modes: P+, P–, DB+, Q+, Q–, S– relative
Direct or indirect
Indexing available
Traps: STOV, BNDV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	0	X	I										

Mode and Displacement

LDX **Load Index. The content of the effective address memory location is loaded into the Index register.**
Memory opcode: 13
Indicators: CCA
Addressing modes: P+, P–, DB+, Q+, Q–, S– relative
Direct or indirect
Indexing available
Traps: BNDV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	1	1	X	I										

Mode and Displacement

STOR **Store TOS into memory. The content of the TOS is stored into the effective address memory location, and is then deleted from the stack.**
Memory opcode: 05, bit 6 = 1
Indicators: unaffected
Addressing modes: DB+, Q+, Q–, S– relative
Direct or indirect
Indexing available
Traps: STUN, BNDV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	1	X	I	1									

Mode and Displacement

LDPP **Load double from program, positive. The double word contained at P + N is pushed onto the stack.**
Sub-opcode 3: 10
Indicators: CCA
Addressing mode: P+ relative
Traps: STOV, BNDV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	1	0	0	0								

P+ Displacement

LDPN **Load double from program, negative. The double word contained at P – N is pushed onto the stack.**
Sub-opcode 3: 11
Indicators: CCA
Addressing mode: P– relative
Traps: STOV, BNDV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	1	1	0	0	1								

P- Displacement

LDD **Load double. The contents of the effective address memory location (E) and the succeeding location (E + 1) are pushed onto the stack. The content of E, the most significant word, is loaded into B; the content of E + 1, the least significant word, is loaded into A. If indirect addressing is used, the word referenced by the initial address (base + displacement) contains a DB+ relative word address. If indexing is used, the effective address is obtained by adding twice the contents of the Index register to the relative word address.**
Memory opcode: 15, bit 6 = 1
Indicators: CCA
Addressing modes: DB+, Q+, Q–, S– relative
Direct or indirect
(for final indirect: DB+ only)
Doubleword indexing available
Traps: STOV, BNDV

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	X	I	1									

Mode and Displacement

STD Store double. The top two words of the stack are stored into the effective address memory location (E) and the succeeding location (E + 1), and are then deleted from the stack. The content of B, the most significant word, is stored into E; the content of A, the least significant word, is stored into E + 1. If indirect addressing is used, the word referenced by the initial address (base + displacement) contains a DB+ relative word address. If indexing is used, the effective address is obtained by adding twice the contents of the Index register to the relative word address.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	X	I	1	Mode and Displacement								

Memory opcode: 16, bit 6 = 1
Indicators: unaffected
Addressing modes: DB+, Q+, Q–, S– relative
Direct or indirect
(for final indirect: DB+ only)
Doubleword indexing available
Traps: STUN, BNDV

LRA Load relative address. The effective address is computed, then the appropriate base register (PB for P+ or P– addressing or DB for DB+, Q+, Q–, and S– addressing) is subtracted. The resulting relative address is pushed onto the stack.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	1	X	I	Mode and Displacement									

Memory opcode: 17
Indicators: unaffected
Addressing modes: P+, P–, DB+, Q+, Q–, S– relative
Direct or indirect
Indexing available
Traps: STOV, BNDV if indirect

LDB Load byte. The content of the effective byte address memory location is loaded right justified onto the TOS. If indirect addressing is used, the word referenced by the initial address (base + displacement) contains a DB+ relative byte address. If indexing is used, the effective byte address is obtained by adding the positive byte index in the Index register to the relative byte address.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	X	I	0	Mode and Displacement								

Memory opcode: 15, bit 6 = 0
Indicators: CCB
Addressing modes: Byte addressing
DB+, Q+, Q–, S– relative
Direct or indirect
Byte indexing available
Traps: STOV, BNDV

STB Store byte. The right byte (bits 8 through 15) of the TOS is stored into the effective byte address memory location and the TOS is deleted. If indirect addressing is used, the word referenced by the initial address (base + displacement) contains a DB+ relative byte address. If indexing is used, the effective byte address is obtained by adding the positive byte index in the Index register to the relative byte address.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	1	0	X	I	0	Mode and Displacement								

Memory opcode: 16, bit 6 = 0
Indicators: unaffected
Addressing modes: Byte addressing
DB+, Q+, Q–, S– relative
Direct or indirect
Byte indexing available
Traps: STUN, BNDV

INCM **Increment memory. The content of the effective address memory location is incremented by one in integer form.**
Memory opcode: 12, bit 6 = 0
Indicators: CCA, Carry, Overflow
Addressing modes: DB+, Q+, Q−, S− relative
Direct or indirect
Indexing available
Traps: BNDV, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	1	0	X	I	0									

Mode and Displacement

DECM **Decrement memory. The content of the effective address memory location is decremented by one in integer form.**
Memory opcode: 12, bit 6 = 1
Indicators: CCA, Carry, Overflow
Addressing modes: DB+, Q+, Q−, S− relative
Direct or indirect
Indexing available
Traps: BNDV, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	1	0	X	I	1									

Mode and Displacement

ADDM **Add memory to TOS. The content of the effective address memory location is added in integer form to the TOS. The result replaces the operand on the TOS.**
Memory opcode: 07
Indicators: CCA, Carry, Overflow
Addressing modes: P+, P−, DB+, Q+, Q−, S− relative
Direct or indirect
Indexing available
Traps: STUN, BNDV, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	X	I										

Mode and Displacement

SUBM **Subtract memory from TOS. The content of the effective address memory location is subtracted in integer form from the TOS. The result replaces the operand on the TOS.**
Memory opcode: 10
Indicators: CCA, Carry, Overflow
Addressing modes: P+, P−, DB+, Q+, Q−, S− relative
Direct or indirect
Indexing available
Traps: STUN, BNDV, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	X	I										

Mode and Displacement

MPYM **Multiply TOS by memory. The TOS is multiplied in integer form by the content of the effective address memory location. The least significant word of the result replaces the operand on the TOS.**
Memory opcode: 11
Indicators: CCA, Overflow
Addressing modes: P+, P−, DB+, Q+, Q−, S− relative
Direct or indirect
Indexing available
Traps: STUN, STOV, BNDV, ARITH

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	1	X	I										

Mode and Displacement

CMPM **Compare TOS with memory. The Condition Code is set to pattern C as a result of the comparison of the TOS with the content of the effective address location. The TOS is then deleted.**
Memory opcode: 06
Indicators: CCC
Addressing modes: P+, P−, DB+, Q+, Q−, S− relative
Direct or indirect
Indexing available
Traps: STUN

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	0	X	I										

Mode and Displacement

INSTRUCTION COMMENTARY

1 MPYL, MPY, DTST, FIXR, FIXT, LMPY. These six instructions provide for the deletion of the most significant word of a doubleword result. The assumption is that the result of the instruction (e.g., multiplication product) does not require more than 16 bits to represent it. The MPY instruction deletes automatically during execution; the remaining five instructions simply test the result and provide an indication (Carry bit) to note whether or not the low order word fully represents the true result. Thus, for these five, the programmer may choose to insert a delete sequence (see figure 2-1) to delete the high order word if it is insignificant.

For MPYL, DTST, FIXR, FIXT, and LMPY, the Carry bit is cleared if the high order 17 bits are all zeros or all ones. This test ensures that the sign bit of the single-length result will be the same as the sign of the double-length result. If this is not the case, Carry is set, and the most significant word should not be deleted. For MPY, Overflow will be set if the test fails, meaning that MPYL should have been used instead of MPY.

2 DFLT, FLT, FADD, FSUB, FMPY, FDIV, FIXR, FIXT. These eight floating point instructions are rounding or truncation in computing a final result and except for DFLT and FLT, are subject to both overflow and underflow. The following paragraphs explain these conditions as they apply to the HP 3000 Series II Computer System.

Rounding and Truncation. Figure 2-2 illustrates both rounding and truncation. Rounding is a simple matter of adding a "1" to whatever is in bit position 32. For FIXR, the binary point of the fixed point result follows bit position 31. If bit 32 is a "1" (case A in the figure), adding "1" will cause a carry into bit 31, thus incrementing the representable value. If bit 32 is a "0" (case B), adding "1" will not cause a carry, and the representable value is not changed.

Truncation is used only by the FIXT instruction and consists of discarding all fractional bits after computing the effective binary point position. This is shown in the lower part of figure 2-2, which illustrates the case of truncating the decimal number 3.5 to 3. The biased exponent (octal 401) represents an exponent of 1. The fraction, as stored, is .11 which, when combined with the assumed leading 1 gives a resultant mantissa of 1.11. The positive exponent of 1 implies that the effective binary point position is one place to the right. Thus the true binary value represented is 11.1, which is 3.5 in decimal. Therefore, in this case, truncation of the fraction consists of discarding all low order bits from 11 through 31.

Overflow and Underflow. Figure 2-3 illustrates overflow and underflow for the 32-bit floating point instructions. Overflow is caused by these instructions when the computed result (either positive or negative) is too large to be represented. Underflow is caused when the computed result is too small to be represented. The limits are defined in figure 2-3.

When user traps are enabled, an overflow or underflow trap will occur to indicate which type of error resulted. If the traps are not enabled, the Overflow bit will be set on either type of error.

It is possible to reconstruct correct answers from overflow or underflow results. If the exponent and fraction are both zero and there is an underflow, the result should be taken as +/− (depending on sign bit) 2^{-256}. In all other cases, test bit 1 (most significant bit of exponent). If this bit is 0, add 512 (decimal) to the exponent; if it is "1", subtract 512 from the exponent to reconstruct the correct biased exponent.

3 ASL, ASR, LSL, LSR, CSL, CSR. The actions of the six single word shift instructions are shown in figure 2-4. It is assumed that the shift count specified in the argument field of the instruction, is 3 in each case. The before and after conditions of the TOS word are shown for each example.

In the case of arithmetic shifts, the sign bit is always preserved. When shifting left, the bits shifted out of bit 1 (most significant bit next to the sign bit) are lost; zeros are filled into the vacated low order bit positions. When shifting right, the sign bit is copied into the vacated high order bit positions, and bits shifted out of bit 15 (least significant bit) are lost.

In the case of logical shifts, all bits are shifted. Bits are lost out of the high end when shifting left and out of the low end when shifting right. Zeros are filled into the vacated bit positions.

In the case of circular shifts, no bits are lost. Bits shifted out of the high end when shifting left are filled into the

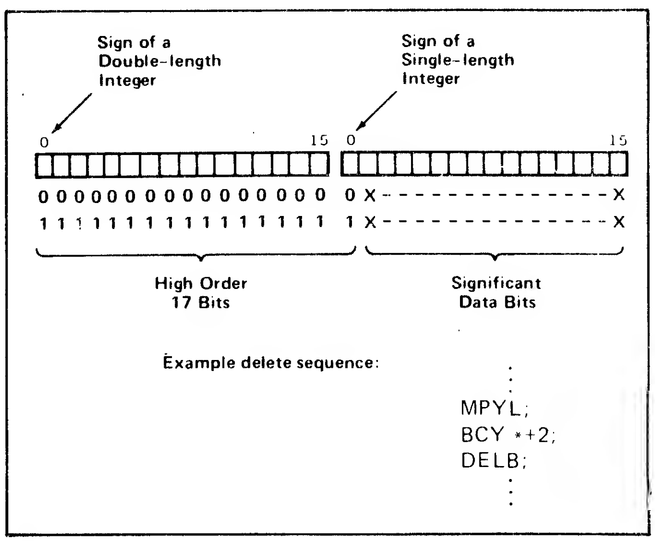


Figure 2-1. Deleting a High Order Word

vacated low order bit positions. When shifting right, bits shifted out of the low end are filled into the vacated high order bit positions.

Note that, for all shift instructions, the number of shifts is determined either by the value specified in the argument field of the instruction or, if X is specified ("1" in bit 4), by adding the argument field value to the Index register contents. This permits the number of shifts to be computed as well as explicitly specified.

All shift instructions except TNSL use the shift count in a modulo 64 manner. Thus if the final shift count is 100 octal (64 decimal), the data is not shifted at all. Furthermore, if the number of shifts equals or exceeds the number of magnitude bits (whether single, double, or triple word), the following will occur: for left arithmetic shifts and all logical shifts, the magnitude will be all-zero; for right arithmetic shifts, all magnitude bits will be the same as the sign bit; for circular shifts, the circular shifting will continue until the specified number of shifts (up to 63) have been achieved.

Except for TNSL (see Instruction Commentary 5) the execution of shift instructions does not alter the content of the Index register.

4 DASL, DASR, DLSL, DLSR, DCSL, DCSR. The actions of the six double word shift instructions are shown in figure 2-5. The shift count, specified in the argument field of the instruction, is assumed to be 3 in each case.

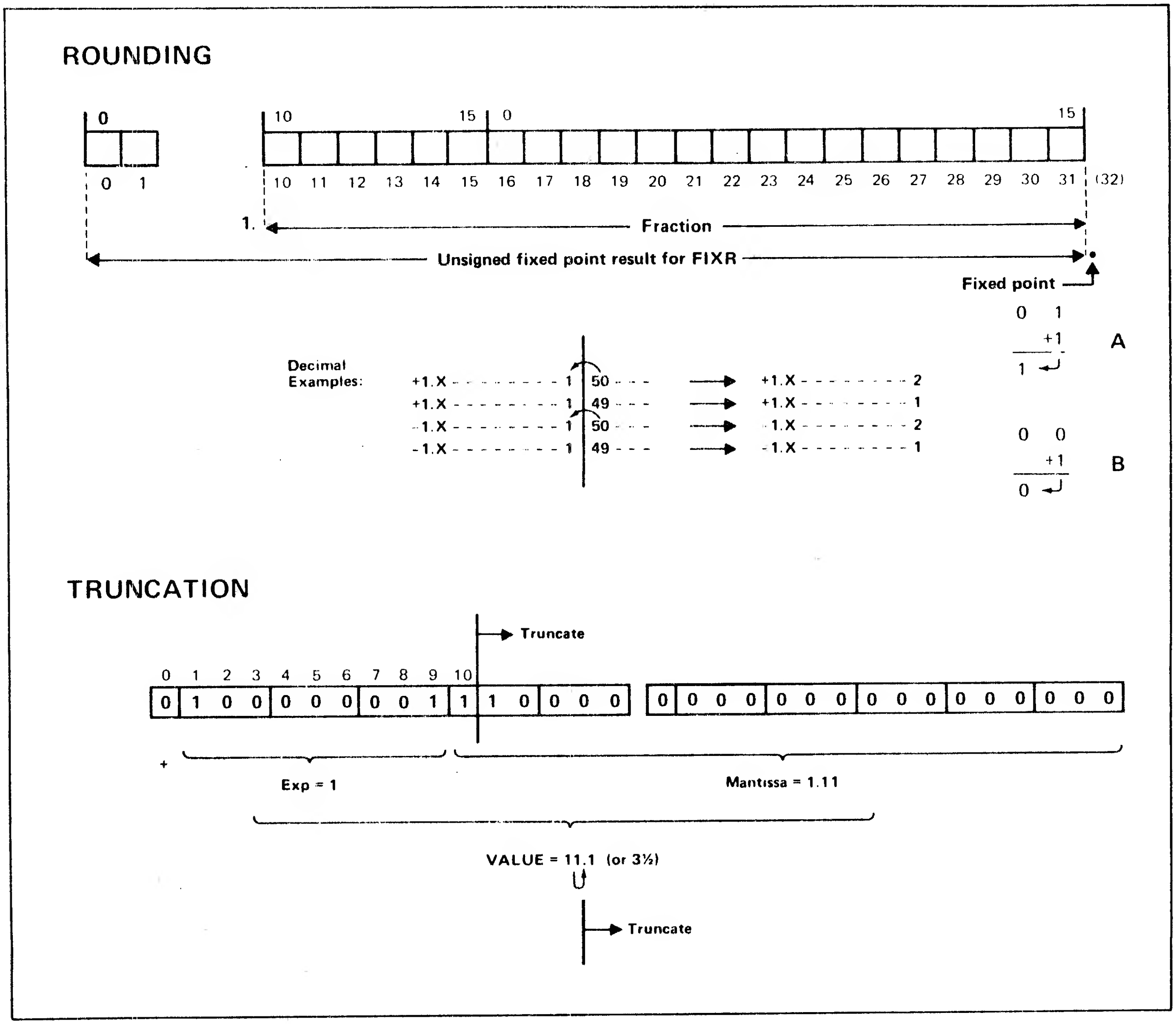


Figure 2-2. Rounding and Truncation

VALUE (Mantissa) Exponent		BINARY REPRESENTATION S	Exponent Decimal	Exponent Binary	Fraction
OVERFLOW (too large to represent)	$+\infty$ / $(2)\ 2^{255}$		⋮ +257 +256		
1.1579 X 10^{77} Decimal (≈)	$(2-2^{-22})\ 2^{255}$	0	+255	111111111	1111111111111111111111
		0	+255	111111111	0000000000000000000000
RANGE OF POSITIVE NUMBERS			+127 +63 +31		
	+1	0	0	100000000	0000000000000000000000
			-32 -64 -128		
(≈) Decimal 8.6362 X 10^{-78}	$(1+2^{-22})\ 2^{-256}$	0	-256	000000000	0000000000000000000001
UNDERFLOW (too small to represent)	$(1)\ 2^{-256}$		-256 -257 ⋮		
ZERO	0	0		000000000	0000000000000000000000
UNDERFLOW (too small to represent)	$(-1)\ 2^{-256}$		⋮ -257 -256		
-8.6362 X 10^{-78} Decimal (≈)	$-(1+2^{-22})\ 2^{-256}$	1	-256	000000000	0000000000000000000001
			-128 -64 -32		
RANGE OF NEGATIVE NUMBERS	-1	1	0	100000000	0000000000000000000000
			+31 +63 +127		
		1	+255	111111111	0000000000000000000000
(≈) Decimal -1.1579 X 10^{77}	$-(2-2^{-22})\ 2^{255}$	1	+255	111111111	1111111111111111111111
OVERFLOW (too large to represent)	$(-2)\ 2^{255}$ / $-\infty$		+256 +257 ⋮		

Figure 2-3. Ranges of 32-Bit Floating Point Numbers

The before and after conditions of the two top words of the stack are given in each example. The TOS contains the least significant half of double word integers, and the second word (B, or TOS-1) contains the most significant half.

Double word arithmetic, logical, and circular shifts are the same as the corresponding single word shifts described above under Instruction Commentary 3 except for the word length. This means that, when shifting left, bits shifted out of the high end of the low order word are filled into the low end of the high order word. When shifting right, bits shifted out of the low end of the high order word are filled into the high end of the low order word. Similarly, on circular shifts, bits shifted out of one end of the double word are filled into the opposite end of the double word.

5 TASL, TASR, TNSL. Figure 2-6 illustrates the actions of the three triple word shift instructions. Two of these, the arithmetic shifts, are the same as the single and double word shift instructions previously described in Instruction Commentaries 3 and 4, except that three words are shifted. The TOS contains the least significant word, B (or TOS-1) contains the middle word, and C (or TOS-2) contains the most significant word.

The TNSL (Triple Normalizing Shift Left) instruction is a special case. Instead of specifying a shift count, TNSL shifts left arithmetically until a "1" is shifted into bit 6 of the most significant word, and the number of shifts is counted in the Index register. The argument field is ignored. Bits 0 through 5 of the most significant word are cleared.

The TNSL instruction clears the Index register before beginning to shift unless X is specified in bit 4 of the instruction. If X is specified, the shift count adds on to the existing contents of the Index register. If bit 6 of C and all lower order bits are zero, a "1" cannot be shifted into bit 6 of C. TNSL initially tests for this condition and, if true, bypasses the shift operations and simply puts 42 into (or adds 42 to) the Index register and does not clear bits 0 through 5 of C. This is the value that would exist if the shifts were actually executed.

The purpose of the TNSL instruction is to normalize a triple word floating point number. Such a number has a 42-bit mantissa consisting of: a leading "1", 38 representable fraction bits, a rounding bit, and two guard bits at the least significant end. TNSL assumes that the number has previously been left-shifted three places in order to include the rounding and guard bits in the least significant word. Thus the leading "1", instead of being assumed to exist in the bit 9 position of C (see figure 2-6) is now moved to the bit 6 position.

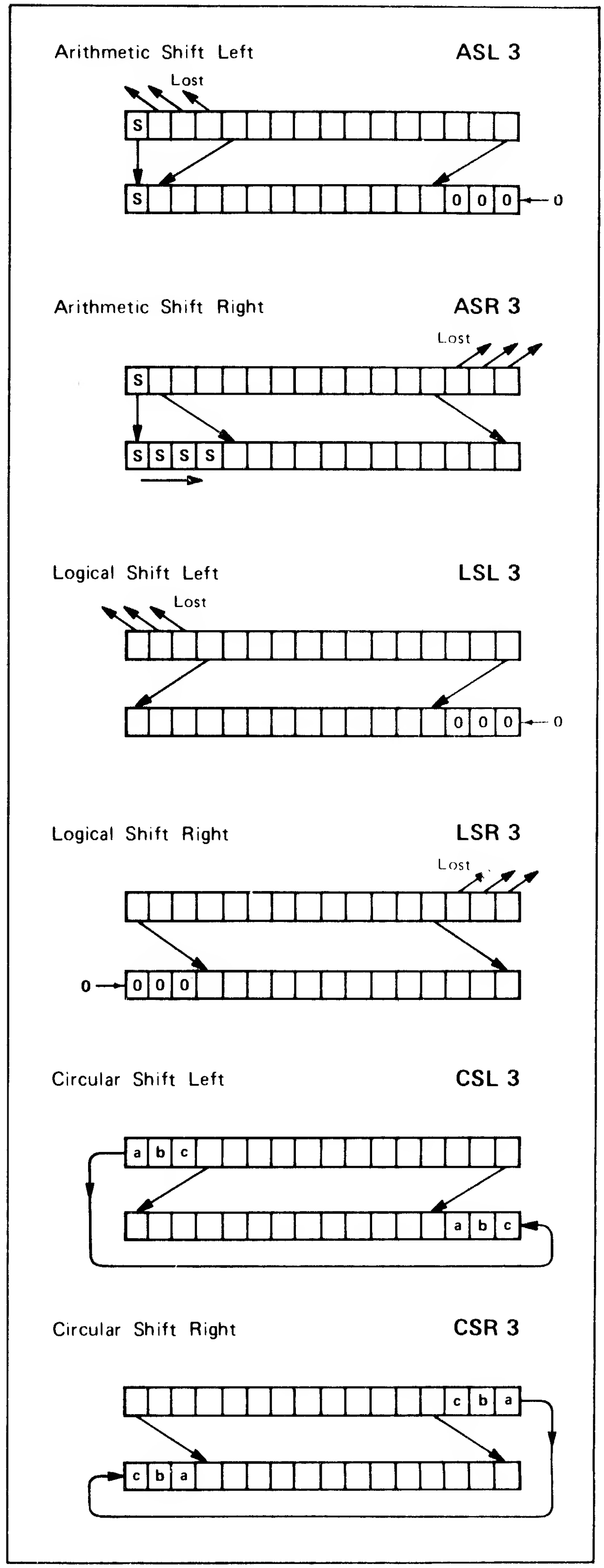


Figure 2-4. Single Word Shifts

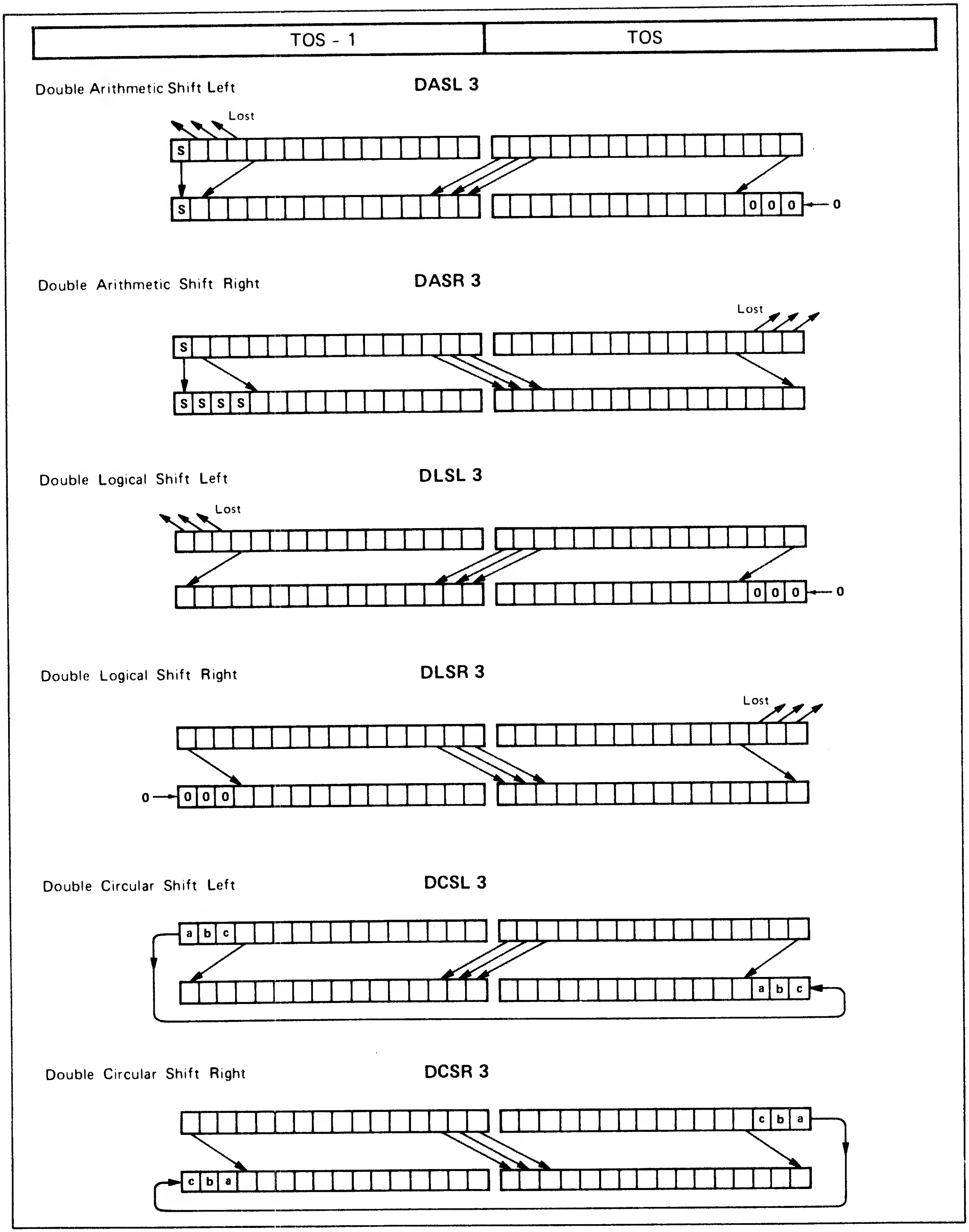


Figure 2-5. Double Word Shifts

6 QASL, QASR. The two quadruple word shift instructions are the same as described previously for the single, double, and triple word shift instructions in Instruction Commentaries 3, 4, and 5, respectively, except that four words are shifted. The TOS (A) contains the least significant word and D contains the most significant word.

7 EXF, DPF. Figure 2-7 compares the operations of EXF and DPF. In the case of EXF, only the TOS word is affected. Assuming values of 2 for J and 8 for K, bits 2 through 9 will be extracted and moved to bits 8 through 15 (i.e., right-justified). Bits 0 through 7, in this example, are filled with zeros. In the case of DPF, the two top words of the stack are affected. The second word of the stack (S−1) is assumed to contain a word that is arbitrarily represented here by the letters "a" through "p". Assuming values of 4 for J and 6 for K, the six least significant bits of the TOS word are deposited into the second word, beginning at bit 4 and ending at bit 9. The remaining bits of the second word are unchanged, and the combined result becomes the new TOS. Note that since the J and K fields each have four bits, they may specify values from 0 through 15 (decimal). The field may wrap around the end of the word; i.e., bit 15 is one bit to the left of bit 0.

8 BR. The P relative mode of BR, the unconditional branch instruction, is a conventional P relative branch except for the indexing capability and the extended displacement range. Bits 8 through 15 are available to specify displacement, which therefore can be up to ±255.

The DB, Q, and S relative mode , however, are unconventional in that they permit indirect branches through the data stack. (It is both illegal and impossible to have a direct branch to the stack; the coding of "01" for bits 5 and 6 encodes the BCC instruction.)

Figure 2-8 shows an example of the S− relative mode. Assume that the instruction in location P specifies the S− relative mode, with a displacement of 4, and indexing. This causes an indirect branch to S−4 in the data stack.

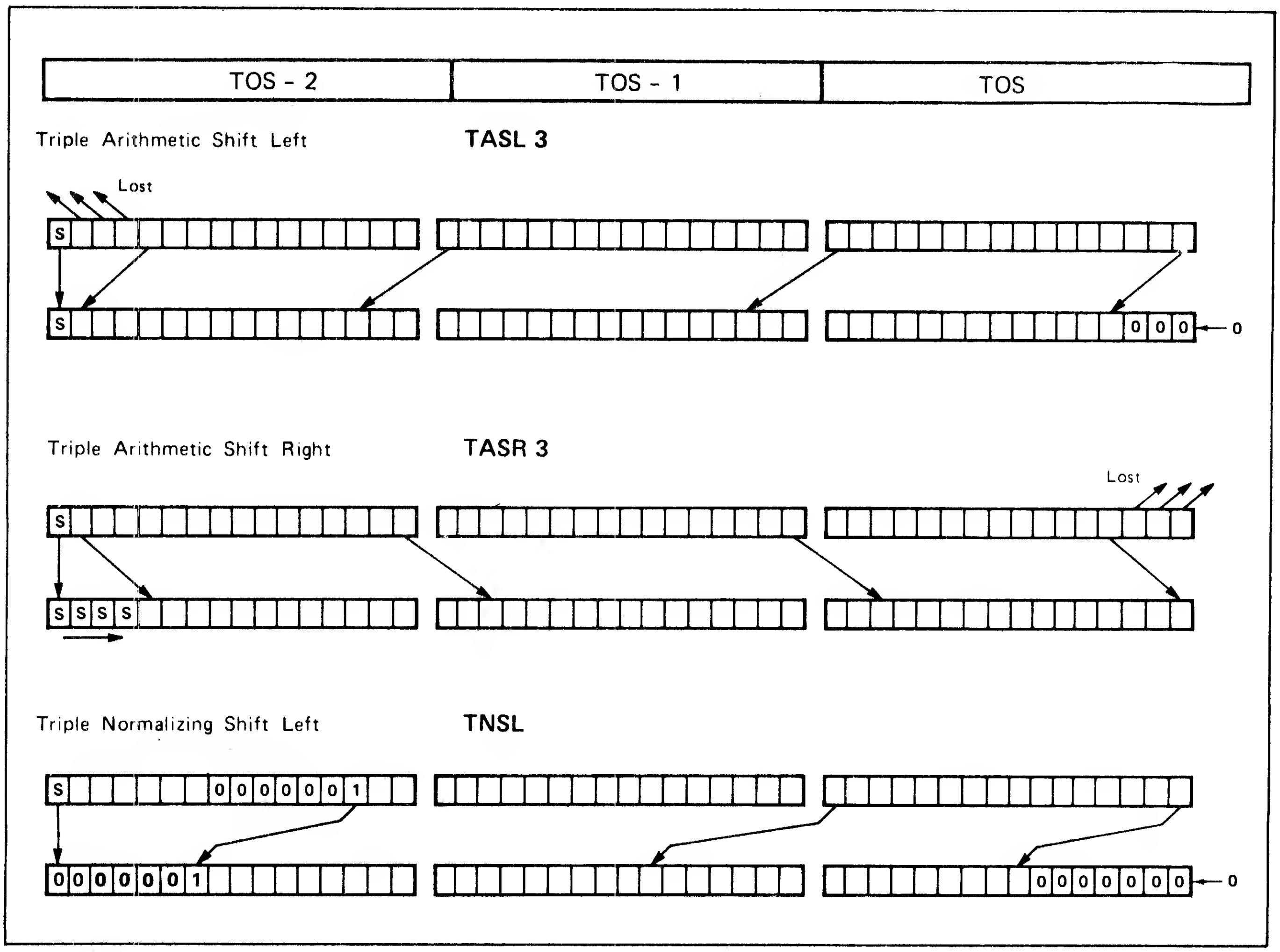


Figure 2-6. Triple Word Shifts

The content of S− 4 is then added to PB, thus pointing at location "a" in the code segment. Since indexing is specified, the value contained in the Index register is also added to the address being computed. Thus the ultimate effective address for the branch (next P) is location "a" displaced by the index value.

Note particularly that the indirect address given in the stack is relative to the program base, PB, not to P as is usually the case. Also note that the displacement is relative to a location in the stack (DB, Q, or S), and that indexing is applied after the indirect addressing has been accomplished.

The displacement range for the DB, Q, and S modes depends on which mode is selected. For DB+, bits 8 through 15 provide a range of 0 through +255. For Q+, bits 9 through 15 provide a range of 0 through +127. For Q− and S−, bits 10 through 15 provide a range of 0 through −63.

9 MOVE, MVB, MVBW, CMPB, SCU, SCW. These six instructions are members of the move group and as such deal with strings of words or bytes. The first three physically move a word or byte string from one block of locations in primary memory to another. The CMPB instruction does not move data but compares data in two complete strings, byte by byte. The last two also do not move data but scan a data string testing the string byte by byte against a test character and a terminal character.

SOURCES. The MOVE, MVB, and CMPB instructions may take source data from either the code segment or the data segment. (For reference purposes, "source" and "target" terminology is retained for CMPB, even though there is no move operation.) If bit 11 of the instruction is a "0", source addresses are PB+ relative — i.e., from the code segment. If bit 11 is a "1", source addresses are DB+ relative — i.e., from the data segment. Figure 2-9 illustrates both cases. Note that the target for either case is in the DB+ area. (Disregard move-direction arrows for CMPB.) Both source and target (MVBW) addresses are DB relative for MVBW, SCU, and SCW. The target need not be "higher" than the source; figure 2-9 shows examples only.

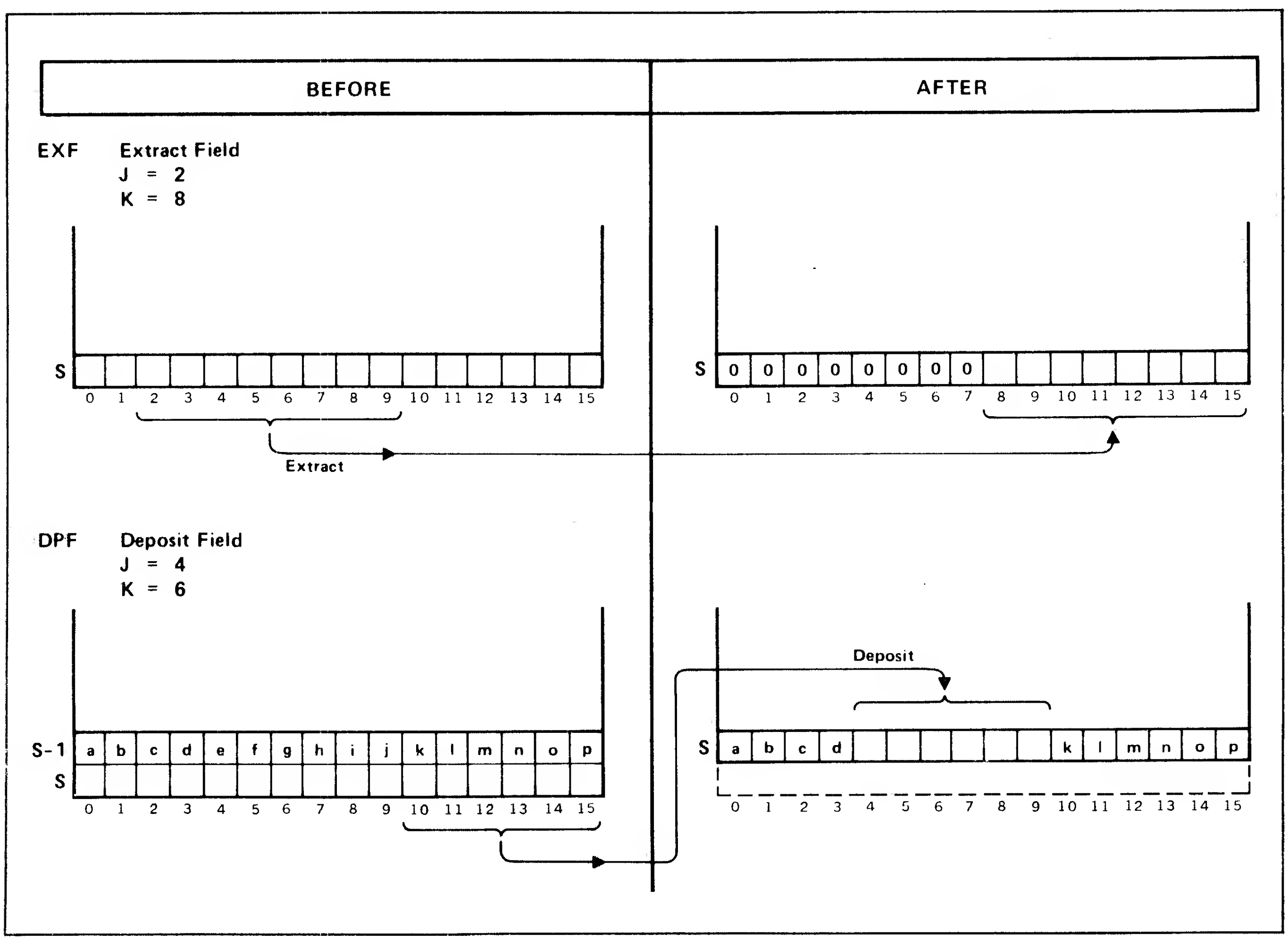


Figure 2-7. EXF and DPF Operation

ASCENDING/DESCENDING ADDRESSES. The MOVE, MVB, and CMPB instructions have the capability of generating ascending or descending addresses for source and target locations. The direction is established by the sign of the count word, which is bit 0 of A, as shown in figure 2-9. If this bit is a "0", the sign is "+", and successive addresses are ascending (B and C incremented). If this bit is a "1" the sign is "−", and successive addresses are descending (B and C decremented). Note the +Count and −Count arrows in figure 2-9. The MVBW instruction uses only ascending addresses; this instruction does not use a count word, and the source and target pointers are in A and B instead of B and C. SCU and SCW also only use ascending addresses; terminal and test characters are in A, the source pointer is in B.

METHOD OF TERMINATION. The MOVE and MVB instructions are terminated only when the word or byte count becomes zero. The MVBW instruction is terminated only when a character of a specified type, either alphabetic or numeric, is encountered. The CMPB instruction has two methods of termination: when the byte count becomes zero, or when any two bytes being compared are unequal. SCU scans until the terminal or test character is found; SCW scans while the string equals the test character.

SPECIAL FEATURES. The MVBW instruction includes an "upshift" bit (bit 13). This bit, when set ("1"), will transpose any lower case source characters to upper case during the transfer. If not set ("0"), the source characters are unaltered by the instruction.

MOVES BEYOND TOS. In the event that the source or target of any move instruction advances into the instruction parameters on the top of the stack or beyond, the parameters (top four if more than four) will not be affected since these values are contained in the top-of-stack registers. The memory locations directly corresponding to these registers will be used for the move (or comparison). However, this situation is normally a software error.

INTERRUPTS. All Move instructions are interruptable and will continue their operation after return from the interrupt. To do this, the count, source, and target addresses are kept updated and deleted from the stack, if specified, only upon completion of the instruction.

10 MVBL, MVLB. These two instructions have many characteristics of the other move instructions described above (Instruction Commentary 8). However, since they move data into or out of the data area between DL and DB, MVBL and MVLB are privileged instructions. The following paragraphs summarize the actions of these two instructions. Refer to figure 2-10.

For MVBL, source data is taken from the DB+ area and the target is in the DL+ area. (A large enough displacement could put the target in the DB+ area.) For MVLB, source data is taken from the DL+ area and the target is in the DB+ area. Addresses for both instructions can be ascending or descending, depending on the state of the count sign. If this bit is a "0", the sign is "+", and succes-

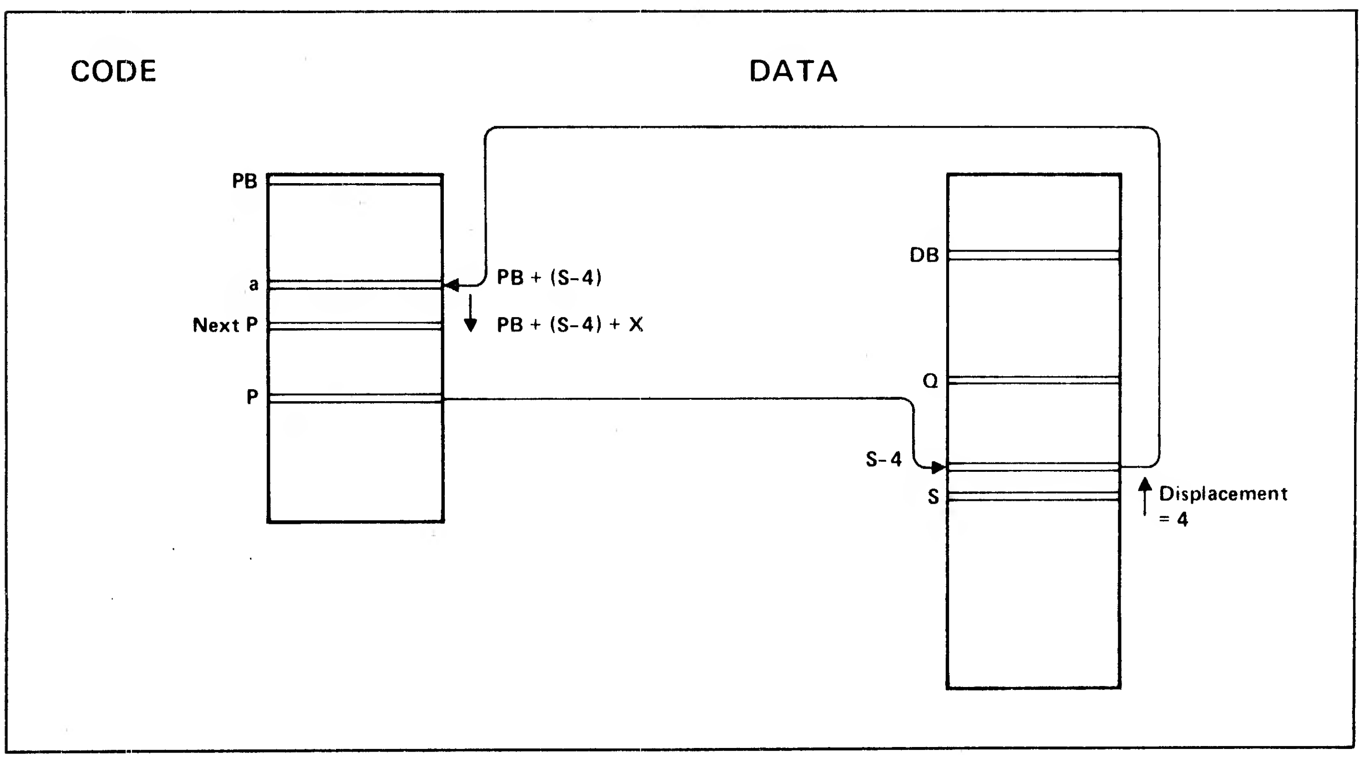


Figure 2-8. Indirect Branch via Stack

sive addresses are ascending (B and C incremented). If this bit is a "1", the sign is "– ", and successive addresses are descending (B and C decremented).

Both MVBL and MVLB are terminated when the word count becomes zero. The comment on "Moves Beyond TOS" under Instruction Commentary 8 also applies to these two instructions.

11 ADDS, SUBS. The reason for the "minus one" when using the TOS content to modify S is to delete the modifying parameter. A typical application of the ADDS instruction is to reserve a block of stack locations for procedure variables. The number of locations so reserved may be either explicitly given in the instruction's operand field, or computed and accessed via the TOS. The effect of the instruction is simply to advance the top-of-stack pointer a given number of locations without specifying any contents. The SUBS instruction, conversely, deletes a specified number of stack locations.

12 SCAL, SXIT. Figure 2-11 illustrates the operations for calling and exiting from a subroutine. Since only local labels may be used, operation is entirely within the current code segment. Assume that the system is executing instructions in the code segment shown in figure 2-11. At some point, P will encounter the "SCAL N" instruction, where N is some value 0 through 255. If the value of N is not 0, e.g., 8, this value will be subtracted from PL (i.e., PL– 8), thus pointing at the ninth cell counting backward from PL. This must be within the Segment Transfer Table, whose first entry is PL– 1. The eighth entry, in this case, contains a local program label (bit 0 = 0), which is a PB relative address pointing to the start of the subroutine. This address is converted to absolute (add to PB) and is loaded into the P-register, while the former value of P, plus one, is stored in the TOS as the return address. However, if N were 0, it would be assumed that the TOS contains the local label (subroutine starting address). This address, then, (made absolute) would be loaded into the P-register, while the former value of P, plus one, replaces the label on the TOS as the return address. In either case, once the P-register has its new address, the location so indicated will be fetched and subroutine execution begins.

The final instruction of the subroutine is SXIT. At this time the return address, pushed onto the stack by SCAL,

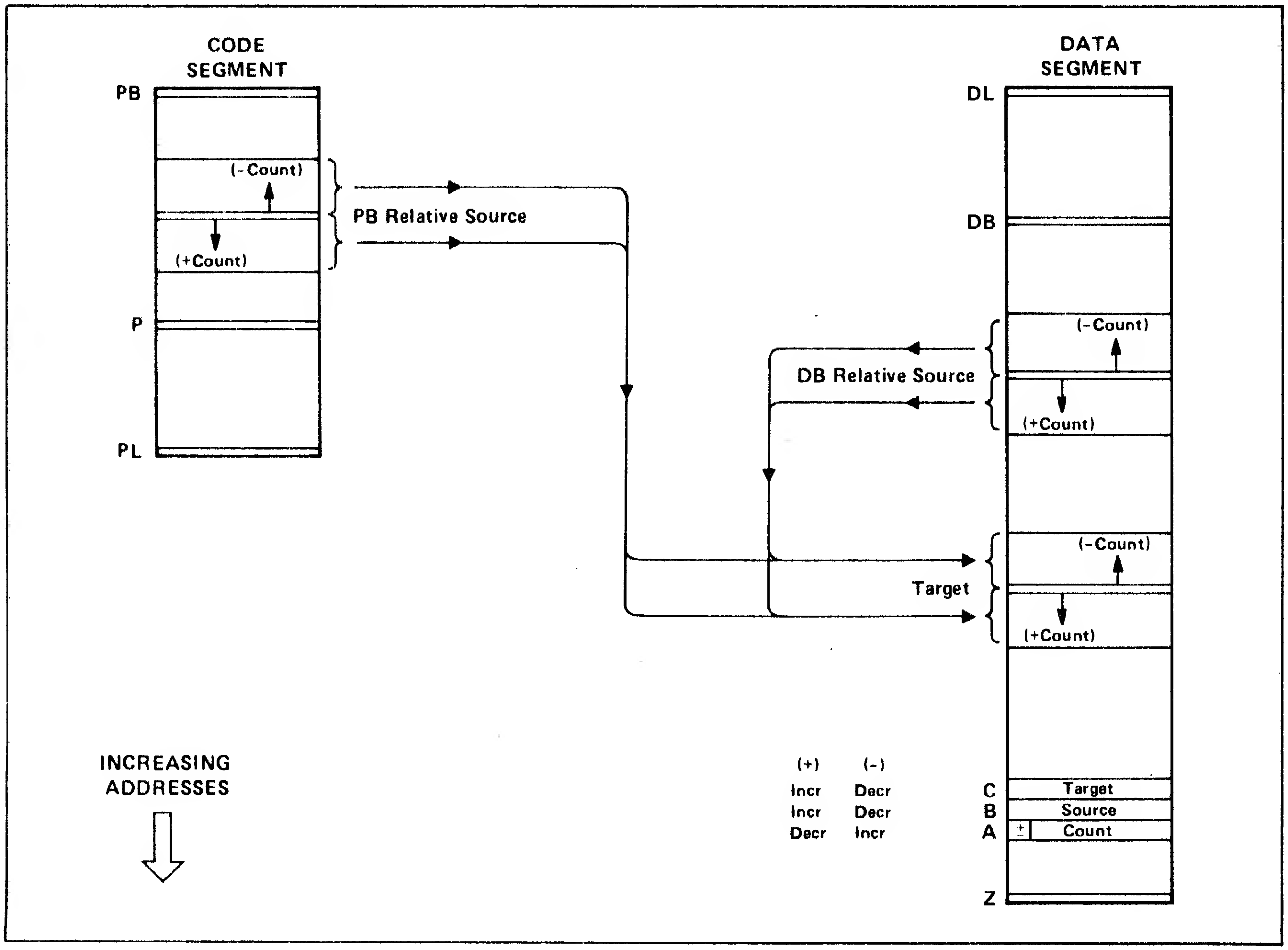


Figure 2-9. Examples of Moves

is assumed to be on the top of the stack. It is the responsibility of the subroutine to provide this condition, which normally means deleting all variables incurred by the subroutine. The SXIT instruction simply takes the address contained in the TOS and puts it in the P-register, thus effecting a return to the calling routine. As a final step, SXIT deletes the TOS, since the return address is no longer needed, and may additionally move S back some number of locations specified by N. This would typically be used for deleting some of the parameters passed to the subroutine.

13 PCAL, EXIT. These two instructions perform basically the same function as the SCAL and SXIT instructions described above (Instruction Commentary 12). That is, to call a routine and return from it to the point where it was called. However, since the routines in the case of PCAL/EXIT may be external to the current segment, possibly not even present in main memory, the operation is somewhat more complex.

The following paragraphs describe the operations of PCAL and EXIT on a step-by-step basis, referring to flowcharts. It will frequently be assumed that the reader has a working knowledge of the intents and purposes of the various steps.

PCAL Sequence. Figure 2-12 illustrates the operations of the PCAL instruction. If the call is within the current segment (local label), only the steps shown on the left side of the diagram are performed. For calls outside the current segment, the steps on the right side are added.

The first step is to fetch the program label. From the PCAL instruction definition, we see that the label can be obtained from one of two places: from the TOS if N is zero, or from PL−N if N is not zero. This operation can be seen in the SCAL operation of figure 2-11, where the label is fetched from either the Segment Transfer Table, at PL−N, or from the TOS.

Thus, referring to figure 2-12, PCAL initially checks N to see if the label is on the TOS. If not (block 1) the label is fetched from PL−N and a check is made to see if that location is actually within the bounds of the Segment Transfer table. (N must be ≤ STTL value in the PL location.) If out of STT bounds, an STT violation is incurred; otherwise, the PCAL sequence continues. If the label is on the TOS (block 2), the label is put into temporary storage in the CPU and S is decremented to delete the label from the stack. At this time, the CPU has the label but does not know whether it is local or external, or if it is valid.

The next step is to place a standard four-word stack marker onto the stack (block 3) and update the Q pointer by loading it with the content of S (block 4). Both Q and S are now pointing at the last word (delta Q) of the new stack marker.

Now the label is checked to see if it is a local label (bit 0 = 0). If it is, the sequence goes directly to block 11 (skip to paragraph starting "Block 8 sets").

If the label is external (bit 0 = 1), bits 8 through 15 are checked to see if the segment number specified is valid. If the segment number does not have an entry in the Code Segment Table, a CST violation is incurred. Otherwise, the PCAL sequence continues. Next, absolute addresses for PB and PB are calculated from the CST entry and loaded into these two registers (block 5).

Block 6 sets the privileged mode in the Status register if the mode bit in the CST entry indicates privileged mode, or if the caller was executing in privileged mode (i.e., if the privileged mode bit in Status already was set). (Although not shown, the Reference bit in the CST is set at this time, for statistical purposes.)

Block 7 stores bits 8 through 15 of the label into bits 8 through 15 of the Status register. This indicates to the system that we are now operating in the called segment.

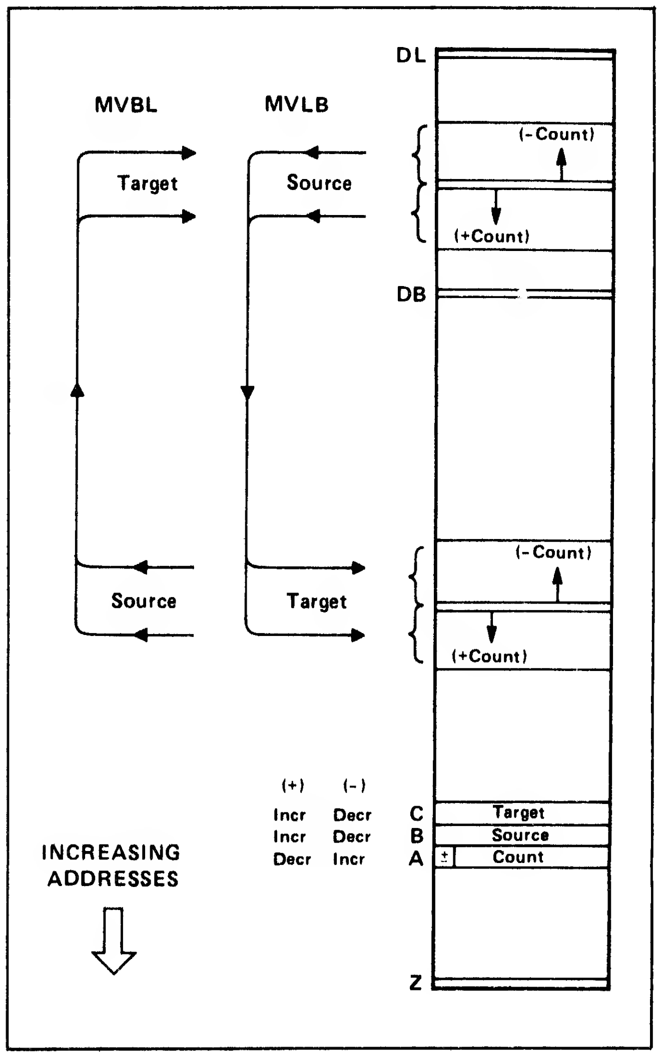


Figure 2-10. Examples of MVBL, MVLB

A check is then made to see if the called segment is absent from main memory. If it is, an absent code segment trap is incurred. A similar check is made for TRACE by checking the CST entry for the called segment.

The next check is to see if bits 1 through 7 of the label are 0. These bits specify which STT entry in the target segment contains the desired local label. Since a value of 0 would point at the STTL word in PL, the value of 0 is specially defined to indicate that P should be set to PB of the called segment, i.e., the local label equals 0. A check is then made to see if the PB entry is callable if it is being called from user mode. Assuming that bits 1 through 7 of the external label are not 0, the value so indicated will point to one entry in the Segment Transfer Table. If it does not (i.e., if the value exceeds the STTL value), or if the entry pointed to is not a local label (i.e., if bit 0 = 1), there will be an STT Violation. But if the label is valid, it is then checked to see if the procedure is callable if being called from user mode by checking bit 1 (must be 0).

Block 8 sets the P-register to the starting address of the procedure. The CPU at this point has a local label, whether it is in the same segment as the PCAL or in a segment external to the calling segment. The value for P is calculated by adding the contents of bits 2 through 15 of the local label to the contents of PB. As a final check, this value for P is checked to see if it is between PB and PL. The resultant absolute value is then loaded into the P-register, and the location so indicated is fetched and execution of the procedure begins.

EXIT Sequence. Figure 2-13 illustrates the operation of the EXIT instruction. If the exit is within the current segment only the steps on the left side of the diagram are performed. For returns to another segment the right side is also executed.

The first step (1) is to fetch the 4-word marker pointed to by Q, which was placed on the stack when the current procedure was called. S is set equal to Q, deleting any local storage being used by the current procedure. If the current procedure is executing in user mode, the privileged and external interrupt enable bits in the marker status are compared with the current status to ensure that the user has not modified these in the marker. Then X is restored from the marker.

In step 2, if the current segment and the segment in the marker are the same steps 3 through 6 are omitted, otherwise continue.

Steps 3 and 4 are similar to the equivalent steps in PCAL (figure 2-12).

In step 5, if in user mode the privileged bit in the CST entry for the return segment must be off. (Although not shown, the reference bit in the CST entry is set at this time for statistical purposes.)

An absent code segment trap occurs following step 5 if the return segment is absent. A trace trap occurs in step 6 if

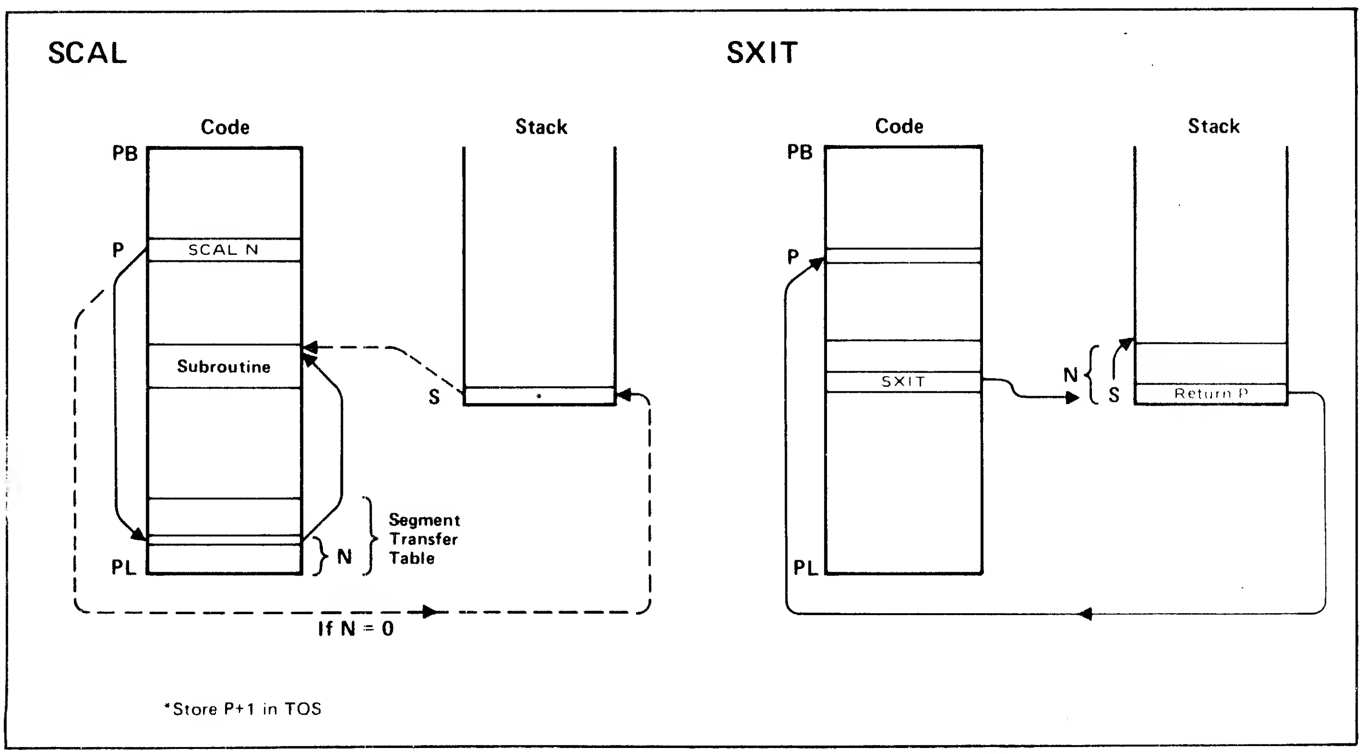


Figure 2-11. Subroutine Call and Exit

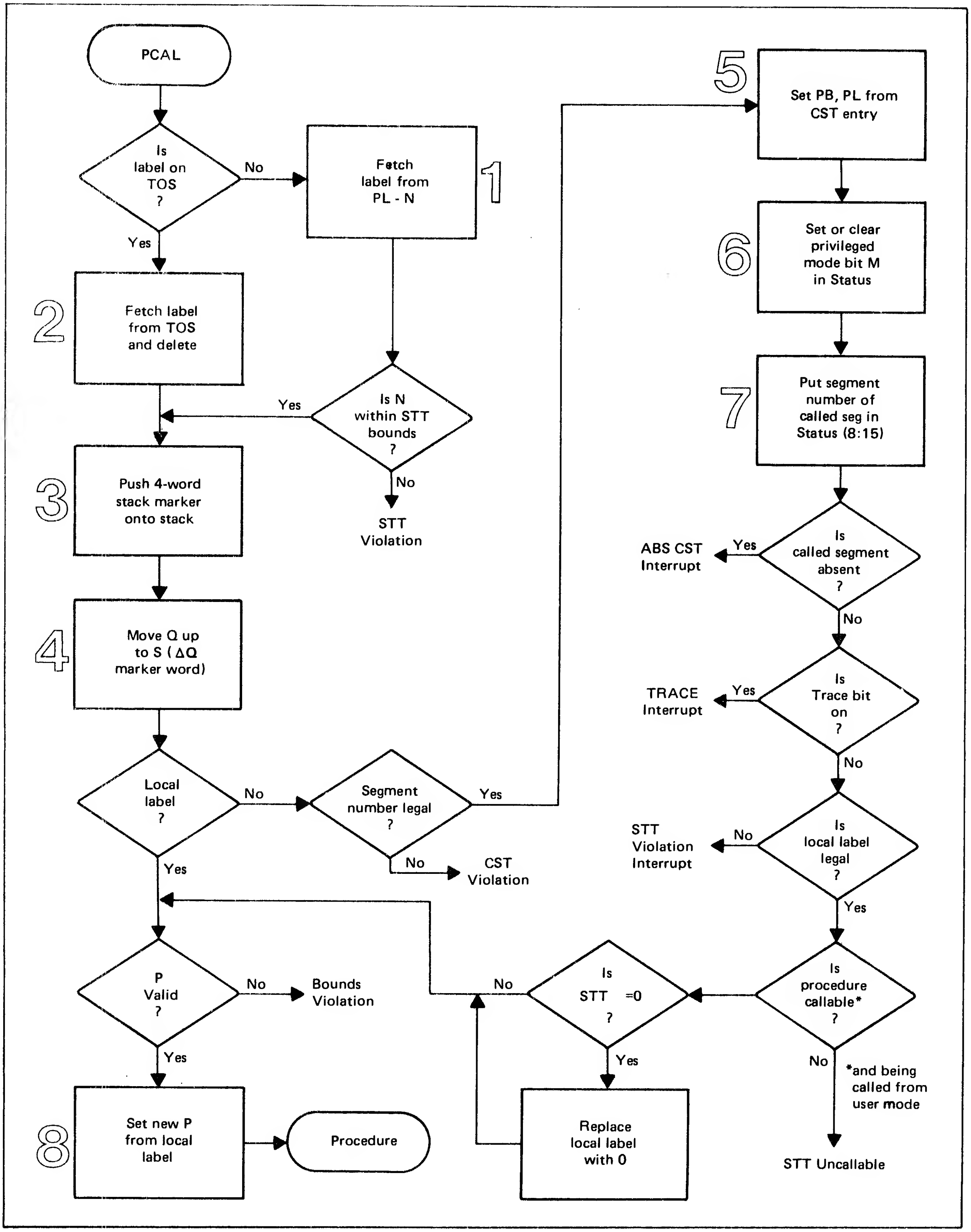


Figure 2-12. PCAL Instruction Flowchart

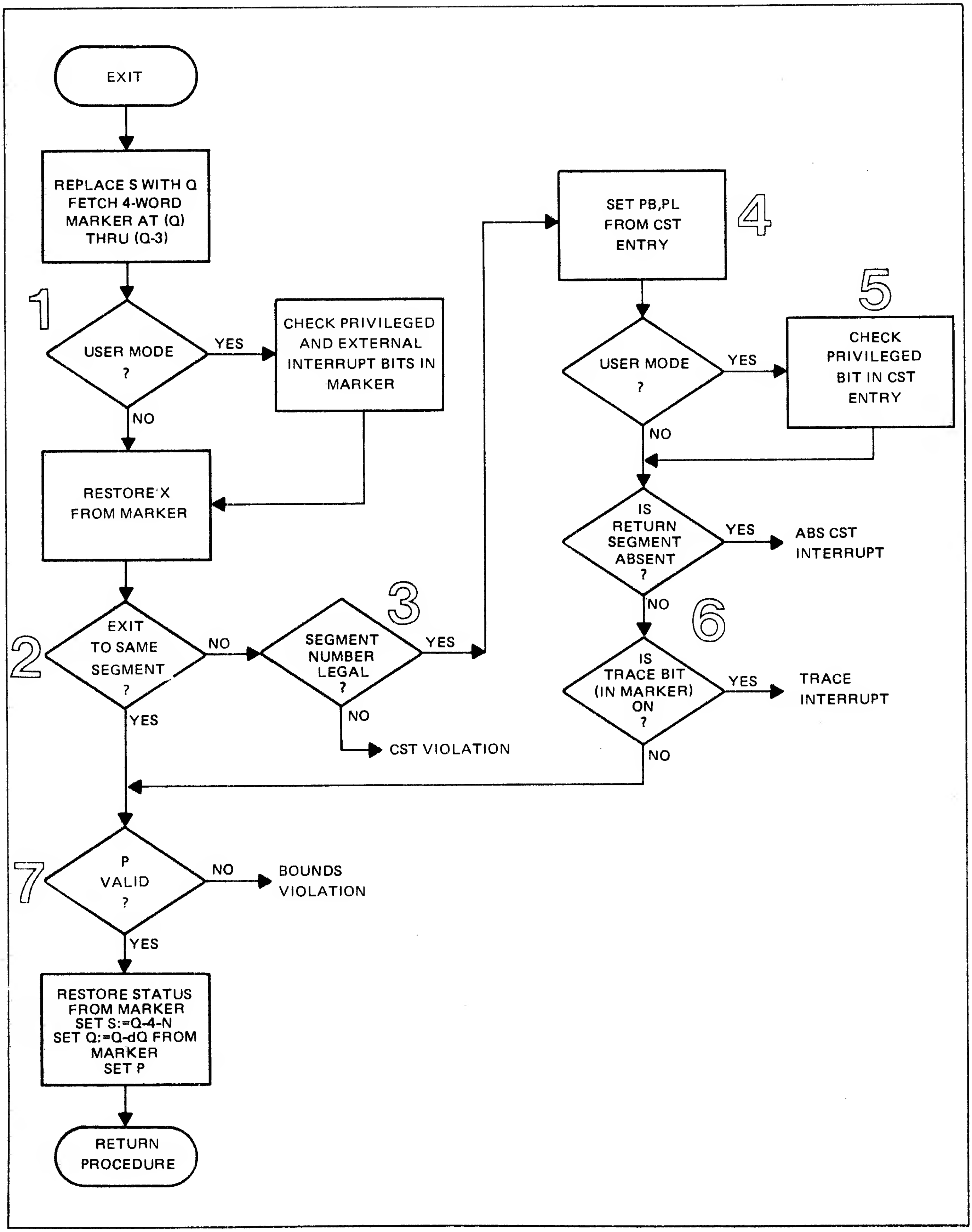


Figure 2-13. Exit Instruction Flowchart

bit 0 of delta P in the marker is set. This bit is normally set by the trace routine which would have been called when the current procedure was entered.

At step 7 return "P = P − delta P" from the marker, must be between PB and PL. The STATUS register is restored from the marker; Q is set pointing to the previous marker, then S is decremented by 4 to delete the marker on the top of the stack and by N (specified in the EXIT instruction) to delete any parameters passed to the procedure being exited. P is set to return P and execution begins within the return procedure.

14 LLBL. The LLBL instruction will convert a local label to external type of it is not already of this type. The conversion is accomplished by forcing bit 0 of the TOS to the "1" state, loading bits 1 through 7 with the value of N (which is the STT entry number), and loading bits 8 through 15 with the corresponding bits of the Status register (i.e., the number of the currently executing code segment).

15 DISP, IXIT, PSDB, PSEB. The Dispatcher, external interrupts, and some internal interrupts execute on the Interrupt Control Stack (ICS). Normally the Dispatch (DISP) instruction is used to enter the Dispatcher and the Interrupt Exit (IXIT) is used to exit from the Dispatcher. Also, when "ICS" type interrupt service routines are entered in response to appropriate events, the instruction IXIT is used to exit from these. The exit may be from the Dispatcher to the process being launched or from interrupt service routines to the interrupted procedure or, in certain cases, to the Dispatcher entry point. The instructions Pseudo Interrupt Disable (PSDB) and Pseudo Interrupt Enable (PSEB) are used to prevent entry to the Dispatcher during critical sections of code.

The instruction DISP causes a transfer to the Dispatcher's entry point unless it is executed while on the ICS or while the Dispatcher is disabled. The Dispatcher is disabled when the Dispatcher Flag is non-zero, (QI− 18) ≠ 0. The address of QI is located at 4 times the CPU number plus 1. Condition code CCE is set when the Dispatcher is entered; the Status register is set as specified for the Dispatcher. The transfer is executed in a manner similar to an ICS interrupt. If a DISP instruction is executed on the ICS or while the Dispatcher is disabled, bit 0 of (QI) is set and CCG is set in the Status register. This bit is checked by those instructions (IXIT and PSEB) which may remove the conditions inhibiting the Dispatcher.

The instruction PSDB increments (QI− 18); PSEB decrements (QI− 18). Starting the Dispatcher is disabled unless this location is zero. Outside the Dispatcher and not on the ICS, a PSEB which decrements (QI− 18) to zero effectively does a DISP instruction if bit 0 of (QI) is set.

Within the dispatcher, a PSEB which decrements (QI− 18) to zero clears (QI), eliminating any pending Start Dispatcher requests. PSDB and PSEB are used at the beginning of the Dispatcher to prevent any interrupts which request a dispatch from causing the first portion of the Dispatcher to be unnecessarily repeated. PSEB instructions which do not transfer to the Dispatcher set CCG in the Status register.

Figure 2-14 is a simplified flowchart of IXIT operation. IXIT operates in one of two manners. The first, (1) in the figure, is by the dispatcher to transfer to a process being launched; the second, (2) through (6), is to exit from ICS interrupt service routines.

If an interrupt service routine is not in segment #1, it is assumed to be an external interrupt routine and a "Reset Interrupt" is sent to the device whose device number is at Q+3. (Q+3) is assumed to be valid in memory, which is normally the case since the device number supplied to external interrupt routines as a parameter is written into memory.

If bit 0 of (Q) is zero, (Q(0)) = 0, then if Q = QI, the return is to be interrupted process (2). Otherwise the return is to a lower priority interrupt which was interrupted (3).

If (Q(0)) = 1 and (QI(0)) = 0, the return is to the Dispatcher which was interrupted (4).

If (Q(0)) = 1 and (QI(0)) = 1, a DISP instruction has been executed and the request to start the Dispatcher is still pending. If (QI− 18) = 0, the Dispatcher is not disabled, QI is cleared, and a transfer is made to the Dispatcher's entry point (5) or (6). It doesn't matter whether a process, Q = QI, or the Dispatcher, Q ≠ QI, was interrupted. If (QI− 18) ≠ 0, starting the Dispatcher is disabled and the DISP request cannot be carried out at this time. Instead IXIT returns to the interrupted Dispatcher, Q ≠ QI (4a), or to the interrupted process, Q = QI (2a). The "Start Dispatcher" request is still pending, (QI(0)) = 1.

16 LLSH. Figure 2-15 illustrates the basic operation of the LLSH instruction. As shown, the top-of-stack (A) contains a 16-bit absolute address within a bank designated by the contents of B. At all times, in successive fashion, this link pointer contains the absolute address of the link word in the segment currently being tested. Location C in the stack is the test word, which would typically be a 16-bit number indicating the size of the segment which is to be brought into memory. Location D is an offset indicating how far the target word is from the link word. Thus as shown, the comparison is between the test word and each target word.

On termination of the instruction, location A of the stack contains the absolute address of the searched-for segment, and a Condition Code of CCE indicates that the search was successful. If the search is not successful, Condition Code CCL or CCG will indicate the cause of termination.

17 XEQ. The reason why the use of a second stack opcode (bits 10 through 15) is illegal is that there is no

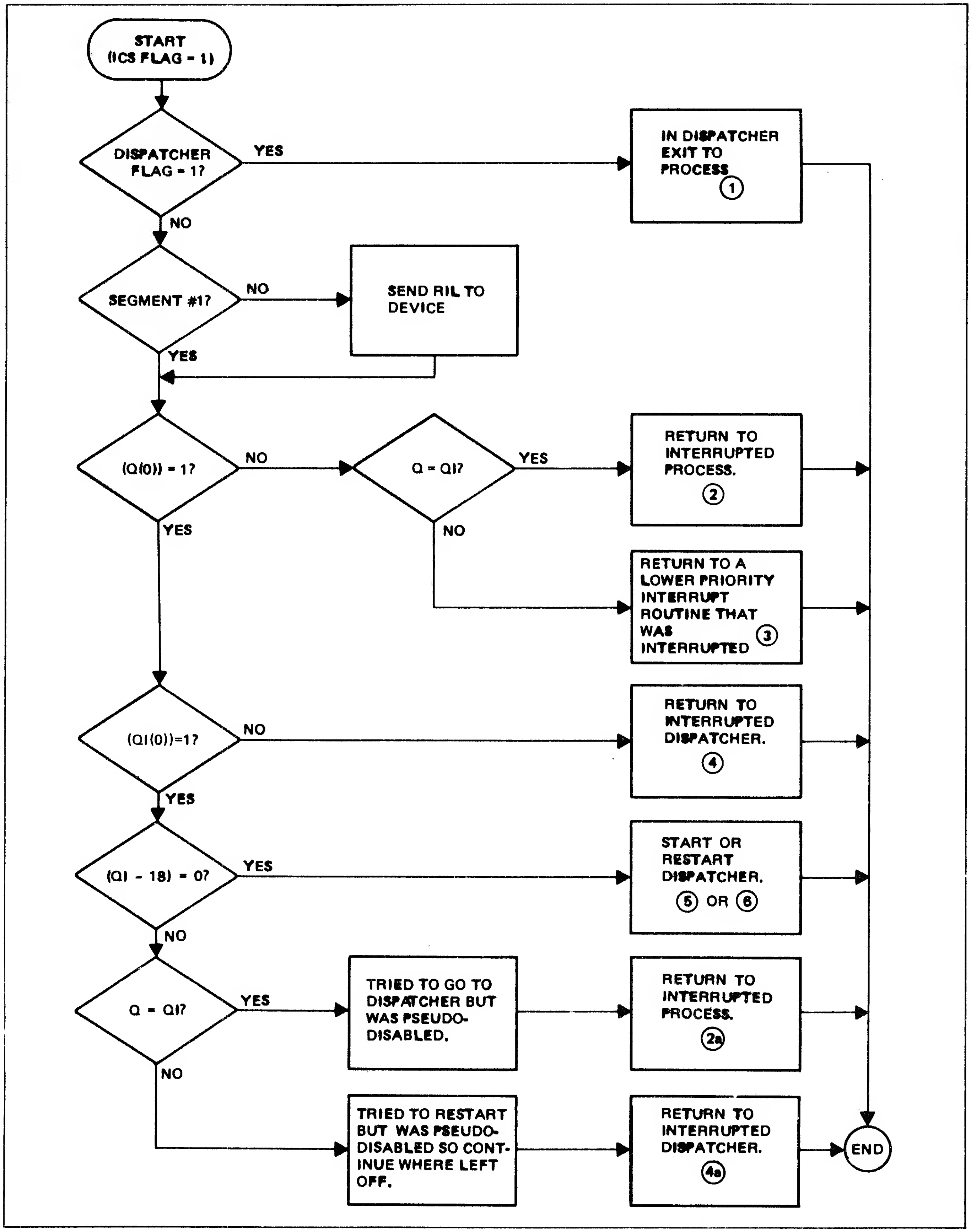


Figure 2-14. IXIT Instruction Flowchart

guarantee that it will be executed. If there should be an interrupt between the execution of the two stack operations, the second opcode will be lost since both came from the data stack rather than a code segment. The interrupt will return to the instruction following the XEQ. However if no interrupt occurs, both stack opcodes will be executed.

18 SIO. There are five I/O instructions in the HP 3000 Command System instruction set. These are:

SIO	Start I/O
RIO	Read I/O
WIO	Write I/O
TIO	Test I/O
CIO	Control I/O

These instructions are fully defined in Section II under the heading "I/O and Interrupt Instructions". The distinction to note here is that the SIO instruction is used in conjunction with an I/O program, and the remaining four are not. That is, the SIO instruction commands a device controller to begin executing its associated I/O program, which effects a block transfer of data between an I/O device and memory. This is termed an "SIO transfer" mode. The other four instructions, on the other hand, transfer only one word per instruction, between the device and the top-of-stack in the CPU.

An SIO type data transfer is initiated by the CPU executing a Start I/O instruction for a particular device. The instruction assumes that there is an I/O program stored in main memory. The hardware I/O system executes the I/O program independently of the CPU. The CPU is then free to continue processing in parallel with the I/O operations.

Figure 2-16 illustrates the order pair format of the double words which are used in I/O programs. The general format is shown at the top of the figure and then the actual format of each of the nine orders is shown beneath. The first word of an order pair is designated as the I/O Command Word, or IOCW, and the second word is designated as the I/O Address Word, or IOAW. The IOAW does not necessarily always contain an address, as the figure shows.

The nine I/O orders are defined as follows:

JUMP. If bit 4 of the IOCW is a "1", a conditional jump of I/O program control is made to the address given by the IOAW at the discretion of the device controller. If bit 4 of the IOCW is a "0", an unconditional jump is made.

RETURN RESIDUE. This causes the residue of the count to be returned to the IOAW. The residue is obtained from the multiplexer or selector channel. Each multiplexer or selector channel has its own count. The count is initialized from the least significant 12 bits of all IOCWs except Return Residue and Set Bank.

SET BANK. This instruction loads the bank register of the multiplexer or selector channel with bits 14 and 15 of its IOAW. The execution of an SIO instruction automatically clears the bank register, therefore, if the data buffer for this device resides in some bank other than 00, the I/O program must contain a SET BANK order prior to a READ or WRITE order.

INTERRUPT. This order pair causes the device controller to set its interrupt request flip-flop and therefore to interrupt the CPU.

END. End of the I/O program. If bit 4 of the IOCW is a "1", the device controller also interrupts the CPU. Returns device status to the IOAW.

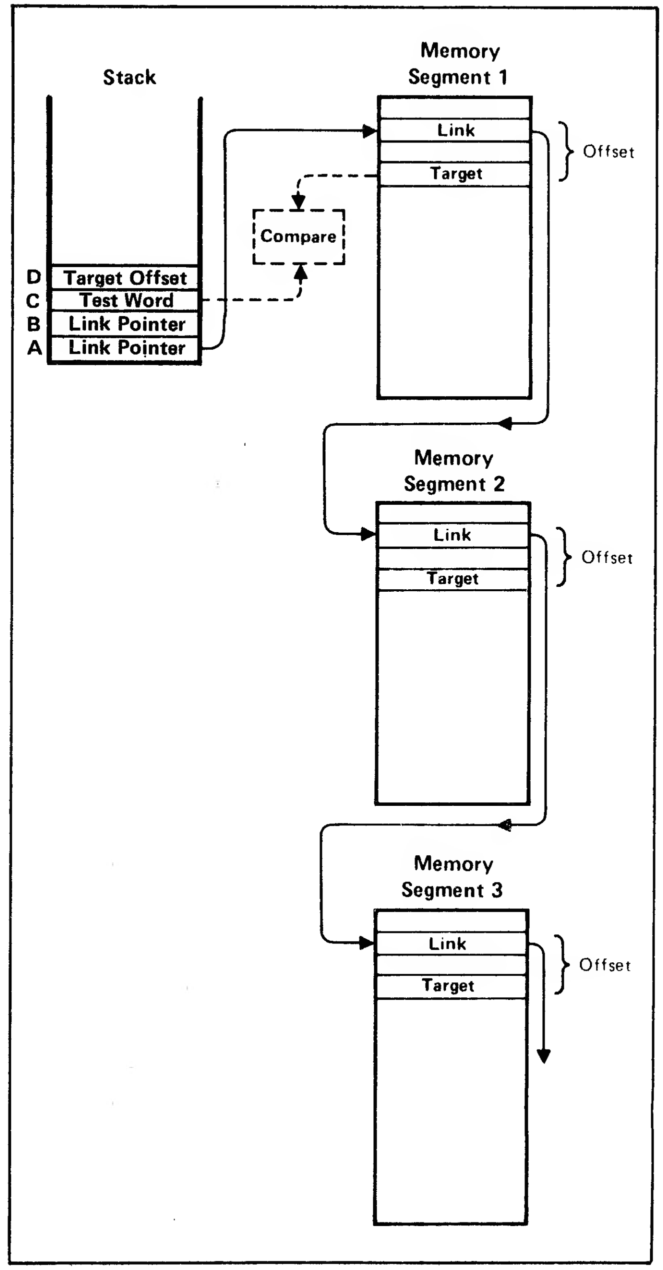


Figure 2-15. LLSH Operation

CONTROL. This causes transfer of a 16-bit control word in the IOAW to the device controller, as well as the 12 low order bits of the IOCW. The IOCW is always available, but a strobe to the device is provided only for Control.

SENSE. This causes transfer of a 16-bit status word from the device controller to the IOAW.

WRITE. This causes "count" words of data to be transferred between main memory and the device, starting at the address given by the IOAW, within a given bank.

READ. This causes "count" words of data to be transferred between the device and main memory, starting at the address given by the IOAW, within a given bank.

Data chaining occurs for Write and Read orders if bit 0 of the IOCW is a "1". This bit may be a "1" for a Write order followed by a Write or for a Read order followed by a Read. This will permit the hardware to treat the counts of each order as a continuous chained count, without reinitializing for each order. The DC bit should be "0" for all other orders.

The count field for Read and Write orders contains the least significant 12 bits of a negative two's complement count value. The count is a word count, independent of the particular recording format (bytes, words, or records). For a Control order, these 12 bits are used for control information in addition to the 16 bits in the IOAW (a total of 28 bits).

TYPICAL I/O PROGRAM OPERATION. Figure 2-17 shows the sequence of operations occurring as the result of an SIO instruction. The sequence is as follows.

1
2 The SIO instruction, decoded by the CPU, fetches the device number given at S-K in the stack, and puts the TOS into the first word of the DRT as the I/O program pointer.

3
4 SIO then loads the device number into the eight least significant bits of the IOP Control Register, and loads an SIO command into bits 1, 2, and 3.

5 The I/O Processor issues the SIO command to the device controller, and execution by the hardware begins. The CPU is now free to continue execution elsewhere.

6 On demand from the multiplexer channel, the I/O Processor obtains the program pointer from the Device Reference Table. (The selector channel obtains the program pointer directly, not via the IOP.) The address is obtained by multiplying the device number by four. The program pointer is the first word of the four-word DRT entry.

7 The program pointer points to the first double word of the I/O program. The pointer is updated to point at each I/O program double word as the program progresses. (The selector channel, to minimize memory fetches, copies the pointer value into a register and updates the pointer internally; the multiplexer channel, however, updates the pointer directly in the DRT.)

8 The sample I/O program is assumed to operate as follows. The first double word contains a CONTROL order which enables the hardware I/O subsystem for this device number. The second double word contains a SET BANK order, which is required if the data buffer for the device resides in some bank other than bank 0 and a Read or Write order is to be processed. The third double word contains a Read order, which causes the subsystem to read a total of 4096 words (or 8192 bytes) into the data buffer whose starting location is given in the IOAW word. Since the data chaining bit is on, the fourth double word is also a Read order, which

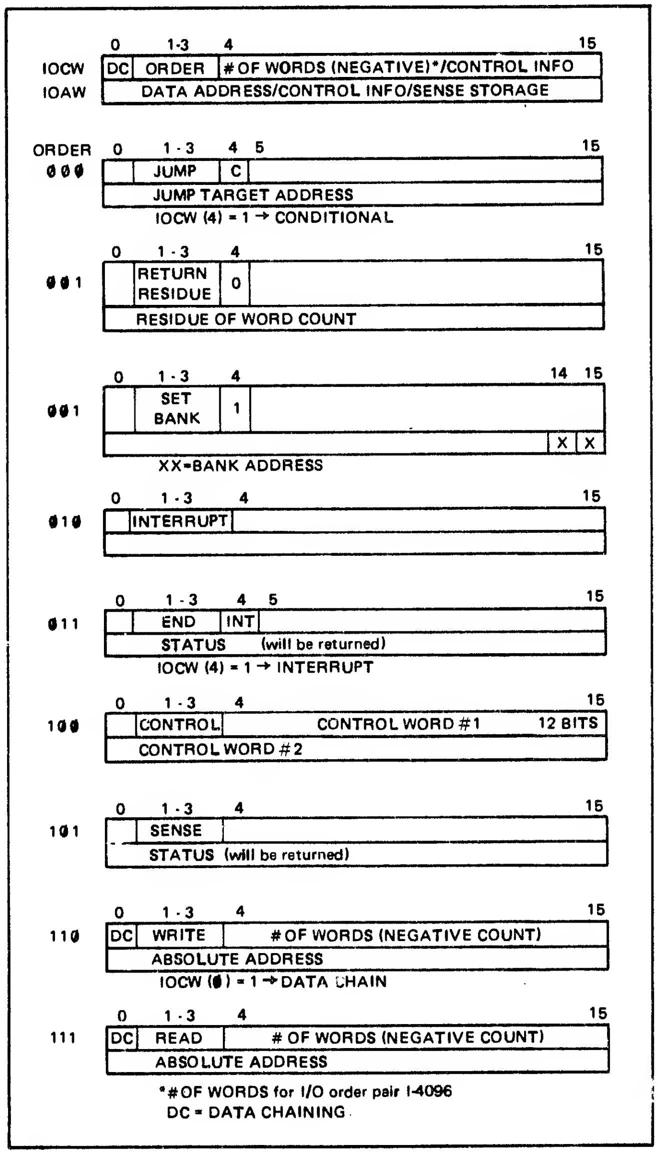


Figure 2-16. I/O Order Pairs

specifies the remaining count required to fulfill the I/O request. (Additional Read orders could be given for larger requests.) The IOAW may specify a buffer area contiguous to the first 4096-word buffer if desired, or in another part of memory if a *scatter read* is desired.

When the transfer is complete, the fifth double word, a CONTROL order, turns off this I/O subsystem. The final double word contains an END order, which obtains the result of the transfer (device status) and loads it into the IOAW; the END order then generates an interrupt to inform the software that the transfer is complete.

At the completion of an I/O program, the selector channel returns the current program pointer value to the DRT. The multiplexer does not take any special action since it updates the DRT after each order fetch.

19 TBA, MTBA, TBX, MTBX. These four instructions perform essentially the same function, and that is to provide a simple mechanism for loop repetition, loop counting, and loop exit, all in one instruction. The differences are that:

a. For TBA and MTBA, the variable is located in the stack; for TBX and MTBX the variable is located in the Index register.

b. For TBA and TBX, modification of the variable is assumed to have been done earlier in the loop, whereas MTBA and MTBX automatically modify the variable as part of their execution function.

With these differences understood, one of the instructions may be taken as a typical example for discussion. Figure 2-18 illustrates one use of MTBA, which is to execute the SPL/3000 FOR statement. As shown, the intent is to vary the value I from 1 to 10 while repeating a certain procedure ten times. (The TBA at the beginning is used to test if the loop is to be executed zero times in the general FOR statement.)

In assembly form, three instructions would be used to initialize the stack. The LRA I instruction puts the DB+ displacement for the variable onto the stack (C), and LDI 1 and LDI 10 push the values 1 and 10 (or octal 12) onto the stack to specify the step increment (B) and limit (A) respectively. The loop is then entered. (If the loop control instruction at the end were TBA or TBX, one of the instructions in the loop would add B to the variable.)

The last instruction of the loop is MTBA, which checks to see if the variable has exceeded the limit. If it has not, control is transferred back (four locations in this example) to the beginning of the loop. The range is P ± 255. At the end of the final loop, MTBA increments the variable to 11, thus exceeding the limit and causing the next instruction

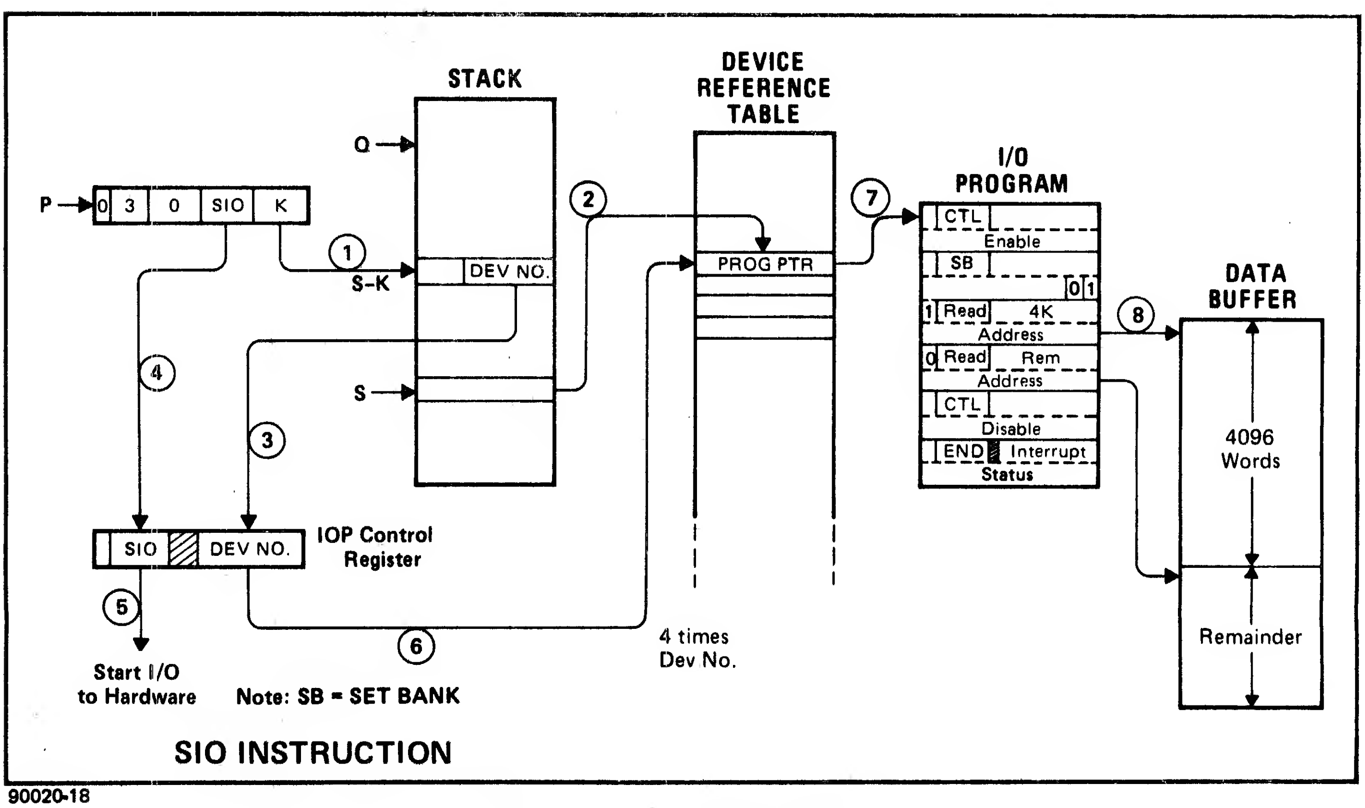


Figure 2-17. Typical I/O Program

in line to be fetched. The three words on the TOS relating to this loop are automatically deleted. The FOR statement has now been executed.

Values for the limit, step, and variable may be negative (two's complement) as well as positive. If step is negative (bit 0 = 1), exit from the loop will occur when the variable becomes smaller (more negative) than the limit, which may be either a positive or a negative number. For MTBA and MTBX, the loop will also be terminated if there is an overflow or underflow when modifying the test variable.

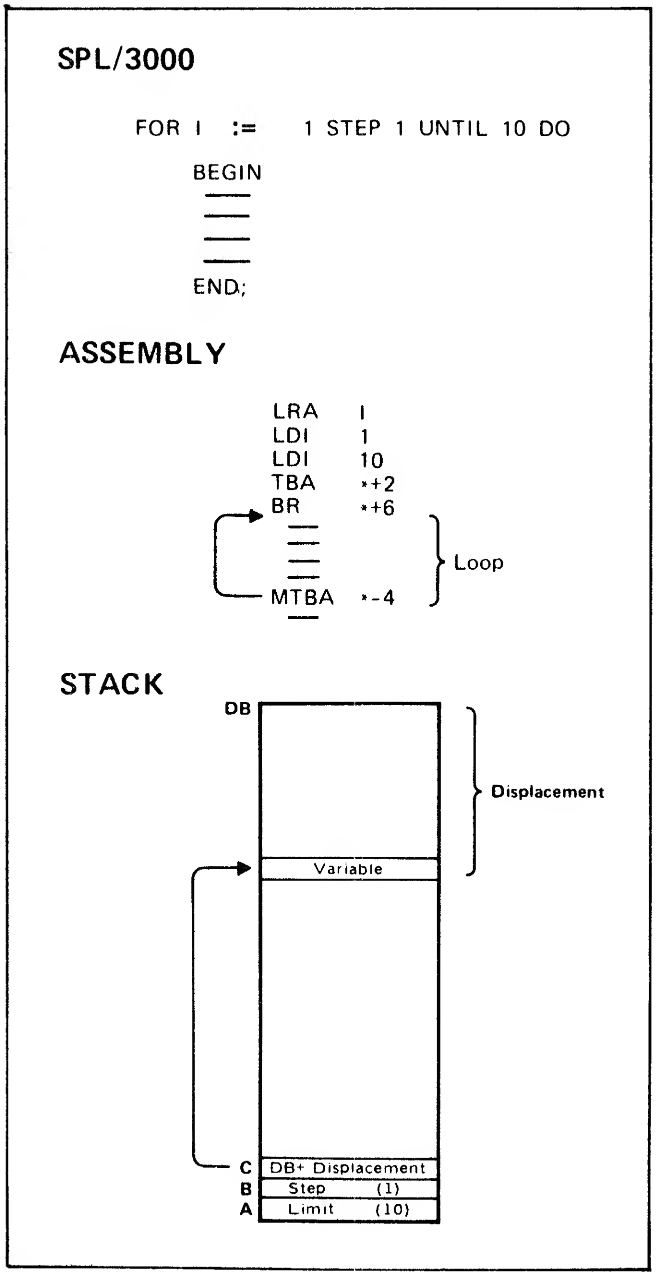


Figure 2-18. Example of Loop Control with MTBA

20 This commentary explains the COLD-LOAD and WARM-START procedures for the Series 30/33 Computer Systems.

It is possible for TP to load and execute a specially constructed program from an I/O device (magnetic tape or disc). This load operation is started by:

1 — pressing the LOAD or START button on the system front panel. (CPU halted).

2 — pressing the LOAD or START keys on the console front panel. (CPU halted).

3 — executing the "STRT" CPU instruction. (CPU running).

Any of these will cause the CPU to enter a special firmware routine which does the following:

1 — if entered from STRT or DUMP instructions, (S)=channel and device number (CDEV); (S-1)= disc head # (HD#).

2 — if entered from either system or console front panel, read CDEV & HD# from the system front panel.

3 — wait 1 sec. (HP 7902 is busy for 1 sec after LOAD/ START/DUMP keys are pressed).

4 — find the size of main memory (# of 128K-byte banks present).

5 — if LOAD/START, initialize all main memory to %030360. (= "HALT %0").

6 — copy the contents of CPU registers to %1401-1422 in main memory: (this is really useful only for DUMP)

```
%1401:= CDEV
    2:= X
    3:= DL
    4:= DB-bank
    5:= DB
    6:= Q
    7:= S
   10:= S-bank
    1:= Z
    2:= STATUS
    3:= PB-bank
    4:= PB
    5:= P
    6:= PL
    7:= CIR
   20:= memory size (# banks)
    1:= system halt #
    2:= ISR (interrupt register)
```

7 — copy the System Interrupt Mask and the LOAD/START/DUMP device DRT entry to %1515-1521: (useful only for DUMP)

```
%1515:= (7)    system interrupt mask
    6:= DRT 0  (CDEV*4)
    7:= DRT 1  (CDEV*4+1)
   20:= DRT 2  (CDEV*4+2)
    1:= DRT 3  (CDEV*4+3)
```

8 — do TOFF; IOCL; INIT(CDEV); IDENTIFY(CDEV); INIT(CDEV); CLEAR.

9 — copy the appropriate channel program to %1423-1505. If the 1st IDENT byte returned by the device is 0, then copy the DISC chan program (for HP 7902,'06,'20,'25); otherwise, copy the TAPE chan program (for HP7970E).

10 — if the 1st IDENT byte was 0 (DISC) and the operation is LOAD/START, then modify the chan program SEEK command to seek to sector 2. Otherwise (DUMP, CDEV is DISC), the seek will be to sector 3.

10 -- if the IDENT code was 0 (DISC) and the operation is

LOAD/START, then modify the chan program SEEK command
to seek to sector 2. Otherwise (DUMP, CDEV is DISC),
the seek will be to sector 3.

11 — setup a DRT for CDEV; do SIOP(CDEV).

The channel program will load 256 bytes from the DISC or TAPE. Those bytes are assumed to be a bootstrap channel program (boot) which will actually load the CPU program to be executed. It is further assumed that the first byte will be a checksum of the rest:
checksum := (sum of 127 bytes) + %123456.
The boot is loaded to %7100 for LOAD, %1530 for DUMP.
If the chan program does not end within 25 sec, there will be a SYSTEM HALT #6 (timeout).

12 — when the chan program halts, compute & check the boot checksum. A checksum error will cause a SYS HALT #7.

13 — if LOAD/START then do SIOP(%7101,CDEV);
if DUMP then do SIOP(%1531,CDEV).
(execute the boot, loaded at %7100 or 1530).
It is assumed that the boot will end with "IN H %0".
If the boot does not end within 25 sec, there will be a SYSTEM HALT #6 (timeout).

14 — when the boot halts, if (CPVA) <> %100000 then SYSHALT #10 (boot aborted or did non-standard halt). Otherwise, setup the ICS and trap to seg 1, STT 44.

```
TP LOAD/DUMP Chan program for DISC

%1423:    1000   WAIT               <<wait for 1st ppoll>>
    4:       0

    5:    2010   WR 10,%1476,2      <<set FILEMASK for HP7920>>
    6:       2
    7:       0
   30:       0
    1:    1476

    2:    1000   WAIT
    3:       0

    4:    2010   WR 10,%1503,2      <<issue READ.STATUS command>>
    5:       2
    6:       0
    7:       0
   40:    1503

    1:    1410   RR 10,%1504,4      <<read 4 status bytes>>
    2:       4
    3:       0
    4:       0
    5:    1504

    6:    1000   WAIT
    7:       0

   50:    2010   WR 10,%1477,6      <<issue SEEK command>>
    1:       6
    2:       0
    3:       0
    4:    1477

    5:    1000   WAIT
    6:       0

    7:    2010   WR 10,%1502,2      <<issue READ command>>
   60:       2
    1:       0
    2:       0
    3:    1502

%1464:    1400   RR  0,%1530,400   <<read 128 bytes to %7100>>
    5:     400
    6:       0
    7:       0    <<note:  read to %1530 for DUMP,>>
   70:    1530    <<                %7100 for LOAD.>>

    1:    1000   WAIT
    2:       0

    3:     600   IN H               <<done - interrupt>>
    4:       0

    5:  177777   END
    6:    7405   FILE.MASK
    7:    1000   SEEK command
 1500:       0      >>note:  head= 1 for "split 7906", 0 else>>
    1:       3      <<note:  sector= 3 for DUMP, 2 for LOAD>>
    2:    2400   READ command
    3:    1400   STATUS command
    4:       0   buffer for status bytes
    5:       0   buffer for status bytes
```

```
 TP LOAD/DUMP Chan Program for TAPE   (HP 7970E)

%1423:    2001   WR 1,%1422,1       <<select unit #0>>
    4:       1
    5:       0
    6:   42000
    7:    1424

   30:    1000   WAIT
    1:       0

    2:    2400   DSJ                <<tape needs it>>
    3:       0
    4:       0

    5:    2001   WR 1,%1474,1       <<issue READ command>>
    6:       1
    7:       0
   40:   42000
    1:    1476

    2:    1000   WAIT
    3:       0

    4:    2400   DSJ                <<tape needs it>>
    5:       0
    6:       0

    7:    1400   RB 1,%7100,400     <<read boot. chan prog>>
   50:     400
    1:    2100
    2:  100000
    3:    7100

    4:       0   JUMP *+2           <<done reading...>>
    5:       2

    6:       0   JUMP *-15          <<burst done... do another>>
    7:  177761

   60:    2007   WB 7,%1475,1       <<issue END command>>
    1:       1
    2:       0
    3:   42000
    4:    1477

%1465:    1402   RR 2,%1476,2       <<read XFER COUNT>>
    6:       2
    7:       0
   70:    2000
    1:    1500

    2:    1000   WAIT               <<NOTE: boot must do DSJ>>
    3:       0

    4:     600   IN H               << done >>
    5:       0

    6:      10   READ command
    7:      23   END  command
 1500:       0   BUFFER for XFER count
```

21 This commentary explains the Series 30/33 Computer System Halt. Certain error conditions are irrecoverable, and will cause the CPU to enter the System Halt state. This state is identical to a normal Halt state, except for the way Halt was entered and the value in NIR (NIR=0 for normal halt). The particular cause of any System Halt can be determined by examining NIR (visible in the LED's on the CPU board):

0	normal halt (no error).
1	STT violation in segment 1.
2	code segment absent on the ICS.
3	Segment 1 absent or traced.
4	stack overflow/underflow on the ICS.
5	CSTL=0. (code segment table length)
6	LOAD/STRT/DUMP — chan program timeout.
7	LOAD/STRT/DUMP — bootstrap chan prog checksum error.
10	LOAD/STRT/DUMP — bootstrap chan prog abort.
11	PSEB instruction found (QI-18)<0.

The System Halt state is cleared (NIR:=0) by the System Reset.

EXTENDED INSTRUCTION SET

SECTION III

This section describes each of the six extended-precision floating-point instructions and each of the 12 decimal arithmetic instructions which complement the basic instruction set of the HP 3000 Computer Systems. Instruction Commentary 1 which follows immediately after the instruction descriptions provides additional information on the extended-precision floating-point instructions. Instruction Commentary 2 then provides additional information on the decimal arithmetic instructions.

EXTENDED-PRECISION FLOATING POINT INSTRUCTIONS

EADD Extended-precision floating point add

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0

Stack before execution:
TOS-2, target address
TOS-1, operand-1 address
TOS, operand-2 address

Operand-2 is added to operand-1, and the rounded normalized sum is stored at the target address. If there is no overflow or underflow, the three addresses are deleted from the stack.
Instruction Commentary 1
Indicators: CCA, Overflow indication for overflow or underflow.
Traps: Extended-precision overflow (%10), extended-precision underflow (%11).
STUN, STOV
Addressing mode: DB+ relative

ESUB Extended-precision floating point subtract

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	1

Stack before execution:
TOS-2, target address
TOS-1, operand-1 address
TOS, operand-2 address

Operand-2 is subtracted from operand-1, and the rounded normalized result is stored at the target address. If there is no overflow or underflow, the three addresses are deleted from the stack.
Instruction Commentary 1
Indicators: CCA, Overflow indication for overflow or underflow.
Traps: Extended-precision overflow (%10), extended-precision underflow (%11).
STUN, STOV
Addressing mode: DB+ relative

EMPY **Extended-precision floating point multiply**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	0	0	1	0	1	0

Stack before execution:
TOS-2, target address
TOS-1, operand-1 address
TOS, operand-2 address

Operand-2 is multiplied by operand-1, and the rounded normalized result is stored at the target address. If there is no overflow or underflow, the three addresses are deleted from the stack.

Instruction Commentary 1
Indicators: CCA, Overflow indication for overflow or underflow.
Traps: Extended-precision overflow (%10), extended-precision underflow (%11).
STUN, STOV
Addressing mode: DB+ relative

EDIV **Extended-precision floating point divide**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	0	0	1	0	1	1

Stack before execution:
TOS-2, target address
TOS-1, operand-1 address
TOS, operand-2 address

Operand-2 is divided into operand-1, and the rounded normalized result is stored at the target address. The remainder, if any, is discarded. If there is no overflow and no underflow, and no attempt to divide by 0, the three addresses are deleted from the stack.

Instruction Commentary 1
Indicators: CCA, Overflow indication for overflow, underflow, or divide-by-zero.
Traps: Extended-precision overflow (%10), extended-precision underflow (%11), extended-precision divide-by-zero (%12).
STUN, STOV
Addressing mode: DB+ relative

ENEG **Extended-precision floating point negate**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	0	0	1	1	0	0

Stack before execution:
TOS, source operand address

An algebraic negate is performed on the source operand. The result is stored at the address of the source operand. TOS is deleted from the stack.

Instruction Commentary 1
Indicators: CCA
Traps: STUN
Addressing mode: DB+ relative

ECMP **Extended-precision floating point compare**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	0	0	1	1	0	1

Stack before execution:
TOS-1, operand-1 address
TOS, operand-2 address

Operand-1 is compared with operand-2. Condition Code CCG, CCL, or CCE is set to indicate that operand-1 is

greater than operand-2, operand-1 is less than operand-2, or operand-1 equals operand-2, respectively. The addresses are deleted from the stack.
Instruction Commentary 1
Indicators: CCC
Traps: STUN
Addressing mode: DB+ relative

DECIMAL ARITHMETIC INSTRUCTIONS

CVAD ASCII to decimal conversion

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	1	0	0		0	0	1	0
											SDEC				

Stack before execution:
TOS-3, target byte address
TOS-2, target digit count
TOS-1, source byte address
TOS, source digit count

Source digits in external-decimal are converted to packed-decimal digits. Source digits except for the rightmost in the field must be leading blanks (%040) or %060-%071. The rightmost digit must be one of those in table 3-1. Leading blanks are converted to packed-decimal zeros. Blanks between digits, or between digit and sign, are illegal. An all-blank field converts to an unsigned (absolute) zero target field. An unsigned external-decimal operand produces an unsigned packed-decimal result. If the number of target digits is less than the number of source digits, the source is converted until the target field is filled, producing a left truncated result. In this case, the remaining source digits are not examined for validity. (It is advisable that the source digit count be less than or equal to the target digit count to take full advantage of the digit checking done in this instruction.) If the source digit count is less than the target digit count, left zero fill is placed in the target field. A stack-decrement (SDEC) bit (instruction word bit 11) will either leave the target address and digit count on the stack or delete all parameters, as specified below. Both digit counts must be in the range $0 < n < 28$. If either the source or target digit count is zero, the stacked parameters are deleted in accordance with SDEC, and execution continues with the next instruction.

SDEC = 0, delete 2 source parameters
SDEC = 1, delete all 4 parameters
Instruction Commentary 2
Indicators: CCA, Overflow
Traps: Invalid ASCII digit (%14)
Invalid decimal operand length (%17)
STUN, STOV
Addressing mode: byte addressing, DB+ relative

CVDA Decimal to ASCII conversion

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	1				0	0	1	1

SDEC
Sign Control

Stack before execution:
TOS-2, target byte address
TOS-1, target digit count
TOS, source byte address

The source packed-decimal digits are converted to fill the target field. An unsigned source operand produces an unsigned external-decimal result. SDEC (bit 11) allows leaving the target address and digit count on the stack or deleting all parameters, as specified below. Two options which affect the low-order result byte are coded in instruction word bits 9 and 10. If bit 9 is a 1, the sign of the source is ignored and an unsigned (absolute) low order external-decimal digit is produced (%060-%071), table 3-1). If bit 10 is 1, one of two result signs are produced. If the source sign is negative the result low-order byte is %175 (−0) or %112 to %122 (−1 to −9). Otherwise an unsigned low-order byte (%060 to %071) is produced. If neither bit 9 nor bit 10 is 1, all 30 bytes listed in table 3-1 can be produced depending on the sign of the source. The condition code is set in accordance with the stored result. An unsigned result is considered positive, so only CCG or CCE can be set if instruction word bit 9 is 1. The effect of bit 9 over-rides the effect of bit 10. If the target digit count is zero, the parameters are deleted in accordance with SDEC and execution continues with the next instruction.

SIGN CONTROL = 00, target sign same as source
SIGN CONTROL = 01, target sign negative if source negative, else unsigned
SIGN CONTROL = 10, target unsigned
SIGN CONTROL = 11, target unsigned

SDEC = 0, delete source address
SDEC = 1, delete all 3 parameters

Instruction Commentary 2
Indicators: CCA, Overflow
Traps: Invalid decimal digit (%15)
Invalid decimal operand length (%17)
STUN, STOV
Addressing mode: byte addressing, DB+ relative

CVBD Binary to decimal conversion

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	1	0	0		0	1	0	0

SDEC

TOS-3, target byte address
TOS-2, target digit count
TOS-1, source word address
TOS, source word count

The number of 16-bit two's-complement binary words specified in the source word count is converted to packed-decimal digits and stored in the target field. If the word count is not in the range $0<=n<=6$, a trap occurs. If the target digit count is not in the range $0 \leq n \leq 28$, a trap occurs. After the binary source is converted, leading zeroes are stored until the target field is filled. If the number of digits generated is greater than the target digit count, the partial result is stored and a decimal overflow trap occurs. SDEC (bit 11) allows leaving either the target address and digit count on the stack

or deleting all parameters. If either the target digit or source word count is zero, SDEC is performed and execution continues with the next instruction.

SDEC = 0, delete 2 source parameters
SDEC = 1, delete all 4 parameters
Instruction Commentary 2
Indicators: CCA, Overflow
Traps: Decimal overflow (% 13)
Invalid source word count (% 16)
Invalid decimal operand length (% 17)
STUN, STOV
Addressing mode: DB+ relative word addressing for source, DB+ relative byte addressing for target.

CVDB Decimal to binary conversion

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	1	0	0		0	1	0	1
											SDEC				

Stack before execution:
TOS-2, target word address
TOS-1, source byte address
TOS, source digit count

The number of decimal digits specified in the source digit count is converted to two's complement binary 16-bit words which are stored in the target field. The number of words produced for various source digit counts is as follows:

Source Digit Count	Target Words
1 to 4	1
5 to 9	2
10 to 18	4
19 to 28	6

SDEC (bit 11) allows leaving either the target address on the stack or deleting all parameters. If the source digit count is zero, SDEC is performed and execution continues with the next instruction.

SDEC = 0, delete 2 source parameters
SDEC = 1, delete all 3 parameters
Instruction Commentary 2
Indicators: CCA, Overflow
Traps: Invalid packed-decimal digit (% 15)
Invalid digit count (% 17)
STUN, STOV
Addressing mode: DB+ relative byte addressing for source, DB+ relative word addressing for target.

ADDD Decimal add

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	1	0			1	0	0	1
										SDEC					

Stack before execution:
TOS-3, operand-2 byte address
TOS-2, operand-2 digit count
TOS-1, operand-1 byte address
TOS, operand-1 digit count

The two operands are added and the result is restored in operand-2 field. A decimal overflow occurs if all significant digits of the result do not fit in the operand-2 field.

This results in a trap, and the left-truncated result is stored in operand-2 field.
SDEC = 00, delete no parameters
SDEC = 01, delete operand-1 parameters
SDEC = 10, delete all 4 parameters
Instruction Commentary 2
Indicators: CCA, Overflow
Traps: Decimal overflow (% 13)
Illegal decimal digit (% 15)
Illegal decimal operand length (% 17)
STUN, STOV
Addressing mode: byte addressing, DB+ relative

SUBD **Decimal subtract**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	1	0	SDEC		1	0	1	1

Stack before execution:
TOS-3, operand-2 byte address
TOS-2, operand-2 digit count
TOS-1, operand-1 byte address
TOS, operand-1 digit count

Operand-1 is subtracted from operand-2 and the result is stored into the operand-2 field. If overflow occurs, that is, if the result digits do not fit in the operand-2 field, the left truncated result is stored in operand-2 and a trap occurs.
SDEC = 00, delete no parameters
SDEC = 01, delete operand-1 parameters
SDEC = 10, delete all 4 parameters
Instruction Commentary 2
Indicators: CCA. Overflow
Traps: Decimal overflow (% 13)
Illegal decimal digit (% 15)
Illegal decimal operand length (% 17)
STUN, STOV
Addressing mode: byte addressing, DB+ relative

CMPD **Decimal compare**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	1	0	SDEC		1	0	1	0

Stack before execution:
TOS-3, operand-1 byte address
TOS-2, operand-1 digit count
TOS-1, operand-2 byte address
TOS, operand-2 digit count

Operand-2 is compared to operand-1 and the condition code is set. The operands remain unchanged at their original addresses.
SDEC = 00, delete no parameters
SDEC = 01, delete operand-1 parameters
SDEC = 10, delete all 4 parameters
Instruction Commentary 2
Indicators: CCC, Overflow
Traps: Illegal decimal digit (% 15)
Illegal decimal operand length (% 17)
STUN, STOV
Addressing mode: byte addressing, DB+ relative

SLD **Decimal left shift**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	1	0	SDEC		0	1	1	0

Stack before execution:
TOS-3, operand-2 byte address

TOS-2, operand-2 digit count
TOS-1, operand-1 byte address
TOS, operand-1 digit count

Operand-1 is moved to the operand-2 field with its digits offset to the left of its sign by the shift amount in the low-order 5 bits of the X register. Leading or trailing digits in the result field which are not supplied by the source operand, will be zeroes. Digits shifted out of the operand-2 field are lost, and carry is set to indicate that significant digits were lost.

SDEC = 00, delete no parameters
SDEC = 01, delete operand-1 parameters
SDEC = 10, delete all 4 parameters

Instruction Commentary 2
Indicators: CCA, Carry, Overflow
Traps: Illegal decimal digit (% 15)
Illegal decimal operand length (% 17)
STUN, STOV
Addressing mode: byte addressing, DB+ relative

NSLD Decimal normalizing left shift

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	1				0	1	1	1
										SDEC					

TOS-3, operand-2 byte address
TOS-2, operand-2 digit count
TOS-1, operand-1 byte address
TOS, operand-1 digit count

Operand-1 is moved to the operand-2 field with its digits offset by the shift amount in the low-order 5 bits of the X register. Leading or trailing digits in the result field which are not supplied by the source operand, will be zeroes. If the shift amount is large enough that significant digits of operand-1 would be shifted out of the operand-2 field, the effective shift amount is reduced so operand-1 is left-justified in the operand-2 field. In addition, a number equal to the difference between the specified and actual shift amounts is left in the X register, and carry is set. If the length of the operand-2 field is such that significant digits would be lost even with a shift amount of zero, a decimal overflow trap is reached and no data movement occurs.

SDEC = 00, delete no parameters
SDEC = 01, delete operand-1 parameters
SDEC = 10, delete all 4 parameters

Instruction Commentary 2
Indicators: CCA, Carry, Overflow
Traps: Decimal overflow (% 13)
Illegal decimal digit (% 15)
Illegal decimal operand length (% 17)
STUN, STOV
Addressing mode: byte addressing, DB+ relative

SRD Decimal right shift

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	1				1	0	0	0
										SDEC					

Stack before execution:
TOS-3, operand-2 byte address
TOS-2, operand-2 digit count
TOS-1, operand-1 byte address
TOS, operand-1 digit count

Operand-1 is moved to the operand-2 field with its digits offset to the right relative to its sign, by the shift amount in the low-order 5 bits of the X-register. Digits shifted into the sign are lost. Zeros are shifted in from the left and high order zeros are inserted to fill the operand-2 field, if necessary. Digits shifted or moved out of the operand-2 field are lost.

SDEC = 00, delete no parameters
SDEC = 01, delete operand-1 parameters
SDEC = 10, delete all 4 parameters

Instruction Commentary 2
Indicators: CCA, Overflow
Traps: Illegal decimal digit (% 15)
Illegal decimal operand length (% 17)
STUN, STOV
Addressing mode: byte addressing, DB+ relative

MPYD Decimal multiply

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	1	0	$\underbrace{\quad}_{SDEC}$		1	1	0	0

Stack before execution:
TOS-3, operand-2 byte address
TOS-2, operand-2 digit count
TOS-1, operand-1 byte address

The operand-2 field is replaced by the product of the operand-1 field times the operand-2 field. If the significant digits of the result do not fit into the operand-2 field, an overflow trap occurs. The results stored in this case will be left truncated unless the actual result is greater than 28 digits. If over 28 digits, nothing will be stored.

SDEC = 00, delete no parameters
SDEC = 01, delete operand-1 parameters
SDEC = 10, delete all 4 parameters

Instruction Commentary 2
Indicators: CCA, Overflow
Traps: Decimal overflow (% 13)
Illegal decimal digit (15)
Illegal decimal operand length (% 17)
STUN, STOV
Addressing mode: byte addressing, DB+ relative

DMPY Double logical multiply

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	1

Stack before execution:
TOS-3, low-order end of operand-2
TOS-2, high-order end of operand-2
TOS-1, low-order end of operand-1
TOS, high-order end of operand-1

The two double-word operands on the top of the stack are logically multiplied. The two operands are replaced by the 4-word logical product. Carry is cleared if the high-order 32 bits are all zeros; otherwise, carry is set.

Instruction Commentary 2
Indicators: CCA (as a 2's complement 4-word integer), Carry
Traps: STUN
Addressing mode: byte addressing, DB+ relative

INSTRUCTION COMMENTARY

1 This commentary explains the use of floating-point numbers in the HP 30012A Extended Instruction Set of the computer system. Figure 3-1 shows the data word format. The word consists of 64 bits, made up of four 16-bit words. The four words are stored in adjacent locations in memory.

Floating-point numbers are stored in a binary sign-magnitude format. Bit 0 of word A is the sign (a 0 represents +; a 1 represents −). Bits (10:15) of word A and words A+1, A+2, and A+3 contain the mantissa and bits (1:9) of word A contain the base 2 exponent.

The exponent, stored in bits (1:9) of word A, is biased. To calculate the actual exponent, 256 decimal must be subtracted from the stored exponent. Figure 3-2 gives some examples of exponents in the extended precision floating-point format. As the figure shows, the unbiased exponent can range from −256 to +255.

The binary point of the mantissa is to the left of bit 10 in word A. Each extended precision instruction normalizes the result produced so that there is an assumed 1 to the left of the binary point unless the result is zero. Similarly, data input to computer is placed in normalized form when converted to floating-point format.

When an instruction produces a new mantissa (as the result of addition, subtraction, multiplication, or division), the amount is always rounded before being stored.

The 55-bit mantissa (including the assumed 1) is equivalent to approximately 16.6 decimal digits.

Figure 3-3 shows some examples of floating-point numbers. In the first example, the mantissa is 1 (the assumed 1). The biased exponent is 2^{256}. The unbiased exponent is 2^0, indicating multiplication by 1. The entire number is therefore 1, and it is positive because the sign bit of the mantissa is 0.

The second example in figure 3-3 also represents 1, but this time the amount is negative because the mantissa sign-bit is 1.

In the representation of +2, the biased exponent is 2^{257}, making the unbiased exponent 2^1, which equals 2. The mantissa is 1, and the total amount is 2×1, or 2. The sign is +.

In the next example the biased exponent is 2^{255}, and the unbiased exponent is 2^{-1}. With the mantissa being 1, the total amount is $1/2^1 \times 1$, equal to 0.5.

In the representation of +256, the biased exponent is 2^{264}, and the unbiased exponent is 2^8. $2^8 \times 1$ is 256.

In the last example the unbiased exponent is 2^8, and the binary mantissa is 1.1. Binary 1.1 equals decimal 1.5, making the total amount 384.

A simple method to determine whether a floating-point number is less than 1 is to examine bit 1 of word A. If the bit is 0, the floating-point number has a value less than 1.

Zero in floating-point form is a special case. Because of the assumed 1, there is no exponent of 2 by which the mantissa can be multiplied to yield zero. Therefore, the format for zero has been established as 64 zeroes (zeroes completely filling the 4-word floating-point number). The floating-point representation of zero is shown at the top of figure 3-4.

The special need for representing zero makes it necessary to disallow the numbers $+\ 2^{-256}$ (+ 8.63 . . . X 10^{-78}) and $-\ 2^{-256}$ (− 8.63 . . . X 10^{-78}). (The $-\ 2^{-256}$ may be treated as -2^{-256} or zero if supplied as an operand to the extended precision floating-point of the HP 30012A. As a result, both $+2^{-256}$ and -2^{-256} if generated are treated as underflow cases.)

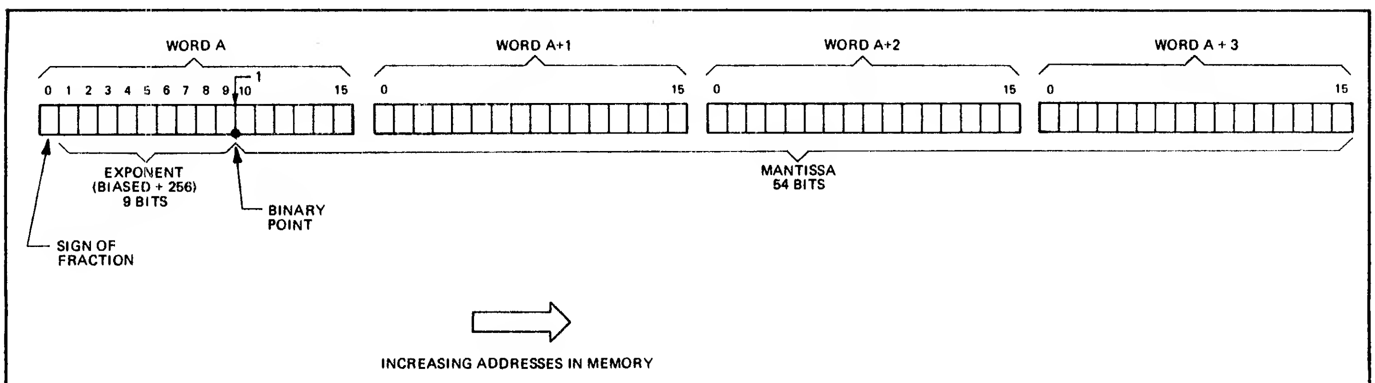


Figure 3-1. Format of Extended-Precision Floating Point Number

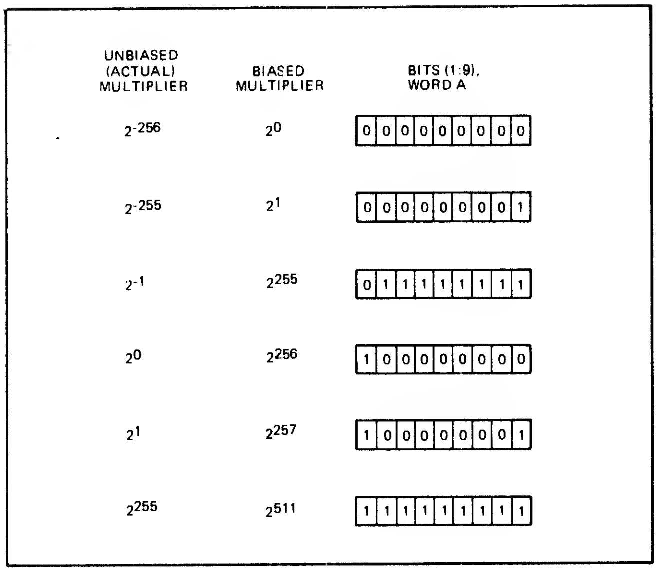


Figure 3-2. Examples of Exponents

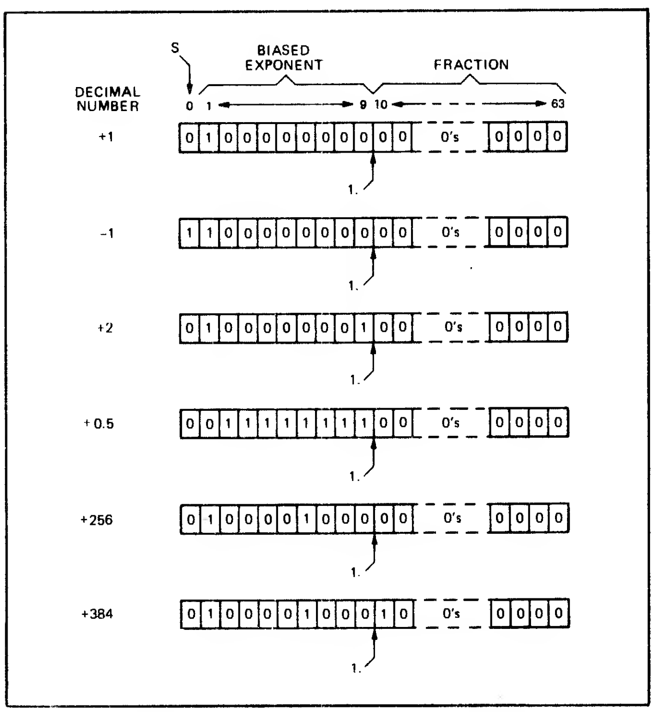


Figure 3-3. Floating Point Numbers and Conversion Formulas

Figure 3-5 shows the range of decimal numbers which can be represented by the 64-bit floating-point number. (The decimal numbers in figure 3-5 are carried to only 11 places.) Intervals G and L in the illustration represent amounts too close to zero to be represented (underflow). If a number in this range results from a floating-point operation, the exponent actually stored is modulo 512 with respect to the true biased exponent. The mantissa is correct. (A number expressed modulo 512 is the remainder which results when 512 is divided into the number.)

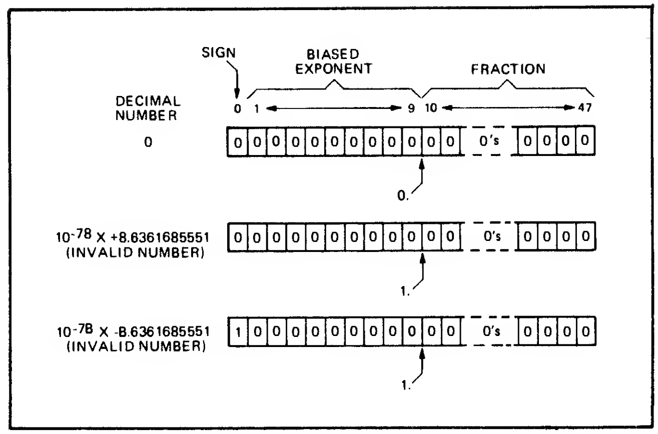


Figure 3-4. Representation of Zero

If the result of a floating-point operation has an absolute magnitude greater than can be represented (overflow), the exponent is expressed modulo 512 and the mantissa is expressed correctly. In figure 3-5, these amounts are above the most positive value shown and below the most negative value shown.

The instructions EADD, ESUB, EMPY, EDIV, and ENEG set condition code A (CCA) in the CPU status register. This indicates whether the result is greater than zero (CCG), equal to zero (CCE), or less than zero (CCL). When this is done, bits 6 and 7 of the CPU status register are respectively set as follows:

- To 00 (CCG) if the result is in interval G or is any other positive quantity, including a positive result too great to be represented.
- To 10 (CCE) if the result is zero.
- To 01 (CCL) if the result is in interval L or is any other negative result whose magnitude is too great to be represented.

The instruction ECMP sets CCG if operand-1 is greater than operand-2, CCE if operand-1 equals operand-2, and CCL if operand-1 is less than operand-2.

The memory address of a 4-word floating-point number is identified by the address of word A. (See figure 3-1.) Word A+ 1, A+ 2, and A+ 3 are in successively higher addresses.

The microcoded extended-precision floating-point instructions are not interruptable. When these instructions are performed by the simulation procedures, interrupts of extended-precision floating-point instructions are recognized in the manner established for the instructions which make up each procedure.

If an EADD, ESUB, EMPY, or EDIV instruction results in an overflow or underflow, CCA is set in the CPU status register, the exponent is modulo 512, and the mantissa is correct as described previously. If the user traps bit (CPU

status register bit 2) is clear, the overflow bit (CPU status register bit 4) is set, the three operand addresses are deleted from the stack, and the next instruction is fetched. If the user traps bit is set, the overflow bit is cleared, the top two addresses are deleted from the stack, and a parameter (000010 octal for overflow, 000011 octal for underflow) is pushed onto the stack. A call is then made to segment #1, STT #25 (decimal).

A divide-by-zero error in EDIV is handled in a similar manner. The dividend is stored as the answer and CCA is set to indicate whether the dividend is positive, zero, or negative. If the user traps bit is clear, the overflow bit is set, the three addresses are deleted from the stack, and the next instruction is fetched. If the user traps bit is set, the overflow bit is cleared, the top two addresses are deleted from the stack and 000012 octal is pushed onto the stack. A call is then made to segment #1, STT #25 (decimal).

If any operands referenced by floating-point instructions reside outside of the stack, a bounds violation trap will occur if executing in user mode. Also, erroneous results will be produced if any of the operand addresses on the stack are part of any of the operands.

2 This commentary explains the use of the decimal arithmetic instructions of the HP 30012A Extended Instruction Set. Most of the instructions use the packed-decimal number format; two use external-decimal number format.

The format of a packed-decimal number is shown in figure 3-6. The left-most byte in the illustration contains the high-order digit. If this digit is in bit positions 4-7, positions 0-3 of the same byte are ignored. (A digit count specifies the number of digits to be recognized.) The characteristics of a packed-decimal number are as follows:

- Each decimal digit is represented in BCD form by four bits.
- The sign is represented by four bits.
- In storage, the four sign bits may be in the following bit positions of a 16-bit word; (12-15) or (4-7). Expressed in different terms, the sign is always in positions (4-7) of an 8-bit byte; the byte is byte 0 or byte 1 of a 16-bit word.

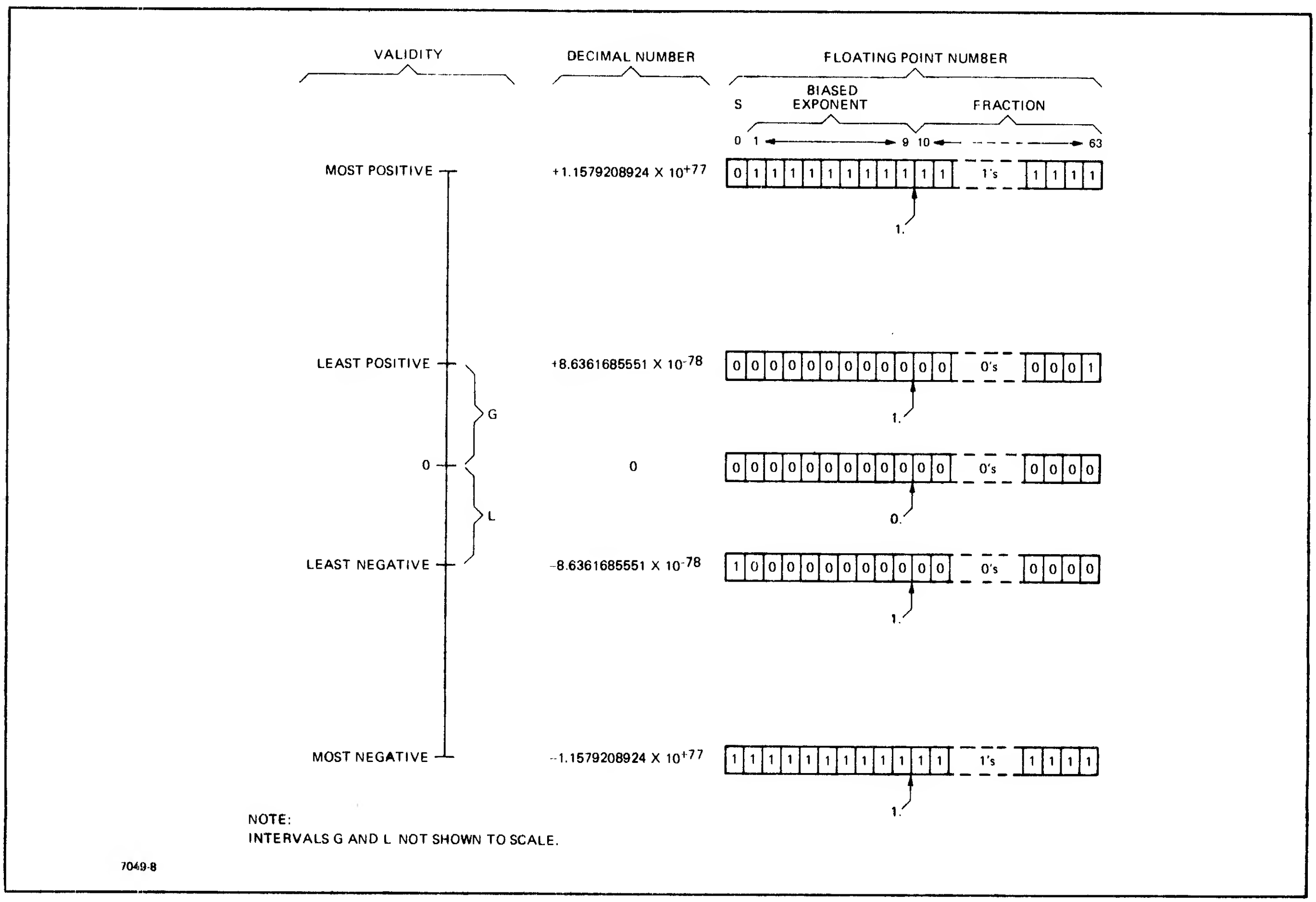


Figure 3-5. Valid Number Range

- If the sign is in (4-7) of a 16-bit word, (8-15) of the same word is not part of the number field and may have any contents.
- Succeeding 4-bit groups to the left of the sign (figure 4-1) contain successively higher-order digits.
- There are no unused bits between the sign and the high-order digit.
- When a packed-decimal number is *source* data for a decimal arithmetic instruction, sign bits 1101 are recognized as minus. All other bit combinations are recognized as plus, except that the CVDA instruction recognizes 1111 as designating an unsigned number.
- When a packed decimal number is the *result* of a decimal arithmetic instruction, sign bits 1100 indicate plus and 1101 indicate minus. There are no unsigned result operands except for the CVAD; the CVAD instruction furnishes 1111 to indicate an unsigned number.
- A leading nonsignificant packed-decimal digit is not modified by any instruction other than CVAD which inserts a zero.

Two decimal arithmetic instructions, CVAD and CVDA, make use of external-decimal numbers. Figure 3-7 shows the format of this type of number.

The characteristics of an external-decimal number are as follows:

- Each digit is represented by eight bits.
- Included in the representation of the low-order digit is an indication of the sign of the number.
- In storage, the low-order digit may be in byte 0 or byte 1 of a 16-bit word. (Byte 0 is the high-order byte of the 16-bit word.)
- If the low-order digit is in byte 0, byte 1 of the same word may have any contents.
- Succeeding bytes to the left of the low-order digit (figure 3-7) contain successively higher-order digits of the numbers.
- There are no unused bytes between the low-order digit and the high-order digit.
- If the high-order digit is in byte 1, byte 0 of the same word may have any contents.
- Table 3-1 shows the low-order digit for positive, negative, and unsigned numbers. The letters A through R in the table, and the braces, are the ASCII equivalent of the 3-digit code shown.

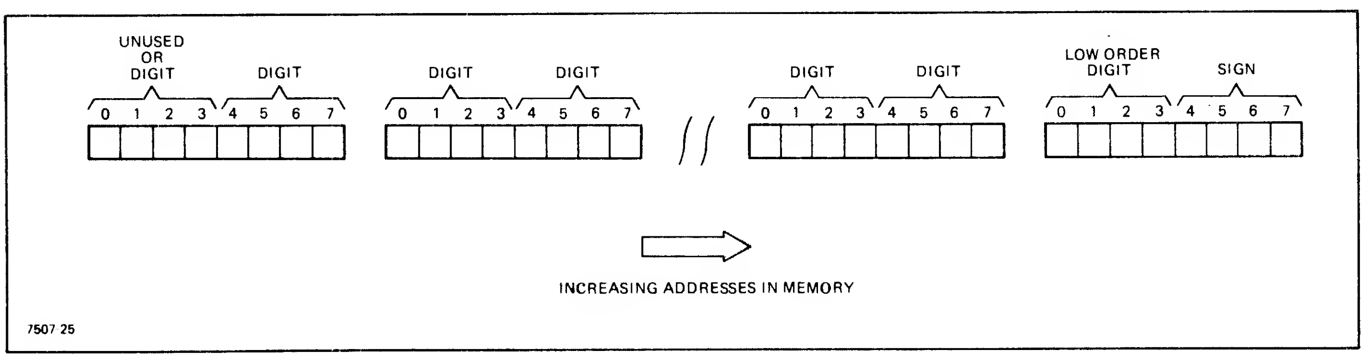


Figure 3-6. Packed-Decimal Format

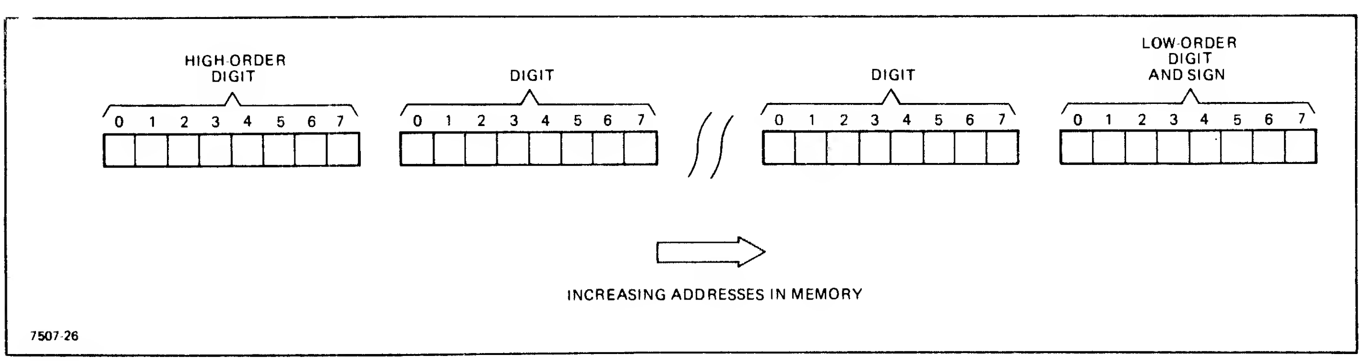
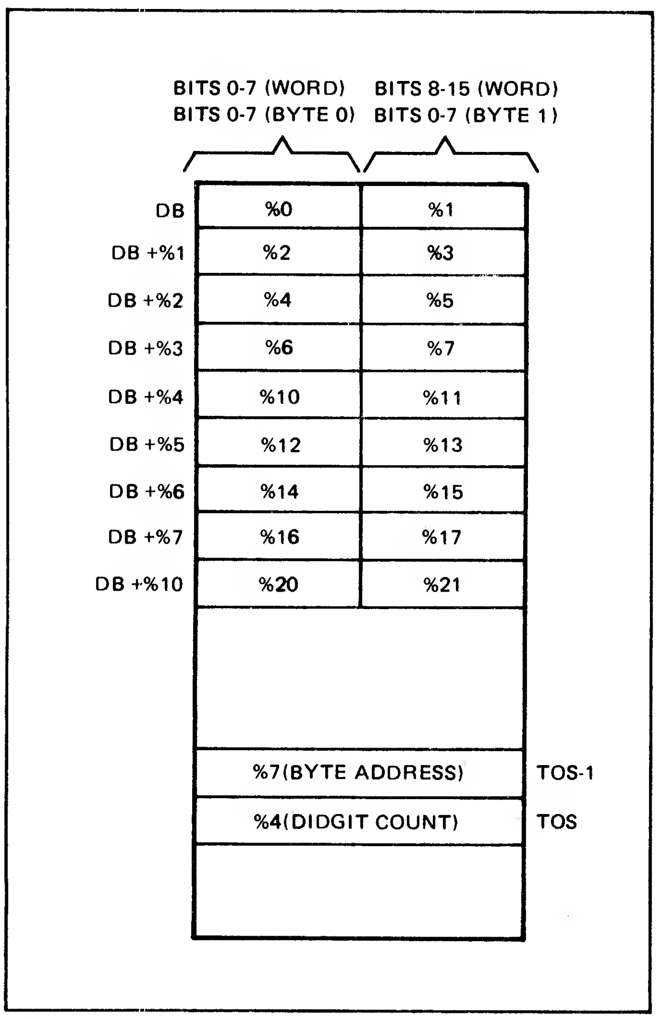


Figure. 3-7. External-Decimal Format

Table 3-1. Low-Order Digits

LOW-ORDER DIGIT, DECIMAL NUMBER	LOW-ORDER DIGIT, EXTERNAL-DECIMAL NUMBER		
	UNSIGNED	POSITIVE	NEGATIVE
0	%060	%173 {	%175 }
1	%061	%101 A	%112 J
2	%062	%102 B	%113 K
3	%063	%103 C	%114 L
4	%064	%104 D	%115 M
5	%065	%105 E	%116 N
6	%066	%106 F	%117 O
7	%067	%107 G	%120 P
8	%070	%110 H	%121 Q
9	%071	%111 I	%122 R



7507-27

Figure 3-8 Typical Packed-Decimal Number in Data Stack

Table 3-2. Error Traps

ERROR	ABORT MESSAGE	TRAP PARAMETER
Packed-decimal overflow	Decimal Overflow	%13
Invalid external-decimal digit	Invalid ASCII digit	%14
Invalid packed-decimal digit	Invalid decimal digit	%15
Source word count >6 or negative	Invalid source word count	%16
Digit count >28 or negative	Invalid decimal operand length	%17

- Digits other than the low-order digit conform with the "UNSIGNED" column of table 3-1.

Each packed-decimal or external-decimal number is stored in the form of a quantity of 8-bit bytes. Each byte occupies bit positions (0-7) or (8-15) of a 16-bit storage location. Successive 16-bit locations are used (if required) to store the entire number. At each end of the number there may be an unused byte in a 16-bit location; this byte may be part of a different number. The storage address of a number is the byte address of the high-order digit.

To illustrate byte addressing, assume that operand-1 of an ADD instruction consists of four significant digits and the byte address is DB+ 7. The sign bits must be in positions 4:7 of a byte, and in this case they are in positions 12:15 of word address DB+ 4 (figure 3-8). This is equivalent to positions 4:7 of byte address 11 (positions 12:15 of word address DB+ 4). The low-order digit is in positions 0:3 of byte address 11 (positions 8:11 of word address DB+ 4). Two digits are in byte address 10. The high-order digit is in positions 4:7 of byte address 7. Positions 0:3 of byte address 7 may have any contents; because the digit count is 4, this part of the byte is ignored.

The decimal arithmetic instructions make checks for improper data as listed in table 3-2. When an incorrect condition is found, one of the following occurs:

- If the User Traps Bit is 0, the Overflow Bit is set to 1. (The User Traps Bit is position 2 of the CPU Status Register. The Overflow Bit is position 4 of the CPU status register.) The stack is decremented as specified.

- If the User Traps bit is 1, a "trap parameter" is pushed on the stack and a call is made to segment #1, STT #25 (decimal). Stack decrementing specified by the instruction is not performed.

Packed-decimal overflow is the condition in which a packed-decimal result has too many significant digits for the specified storage size. (The target digit count is too small.) Except for the NSLD and MPYD instructions, when this occurs the low-order digits of the result are stored; surplus high-order digits are discarded. The NSLD instruction stores nothing in this case. The MPYD in-

struction stores the left truncated result if the full result could be contained in 28 digits; otherwise, it stores nothing.

Two of the error conditions in table 3-2 are for invalid digits. An invalid digit has a bit combination for which there are no provisions in the number system being used.

Decimal arithmetic instructions check the validity of digits as follows:

- In a packed-decimal source operand, all digits are checked.
- In an external-decimal source operand, the digits checked are those specified by the target digit count.

An invalid external-decimal digit is one of the following:

- For the signs a bit combination not shown in table 3-1 unless the field is all blanks.
- If other than a low-order digit, a bit combination not shown in the "UNSIGNED" column of table 3-1. Leading ASCII blanks (%040) are valid; blanks between digits are not valid.

The CVAD instruction checks for invalid external-decimal digits. Only the quantity if source digits specified by the target digit-count are checked by CVAD.

An invalid packed-decimal digit has a value greater than 1001 binary. (In hexadecimal notation, the invalid bit combinations represent the letters A through F.) All decimal arithmetic instructions which use a packed-decimal source operand check each BCD digit for validity; sign bits are not checked.

An error trap occurs if the digit count is greater than 28 or negative. If an instruction uses two digit counts (source and target counts), either count can set the error condition.

An error trap occurs if the source word count is greater than 6 or negative. This error trap is used only by the CVBD instruction, which makes no change at the target address if the trap conditions occurs.

Each decimal arithmetic instruction sets CCA in the CPU Status Register in accordance with the result operand. As an exception, CMPD sets CCC.

If a result is truncated because the target digit count is too small, CCA is set in accordance with the truncated result.

When the stored result is truncated because the target digit count is too small, CCE (operand = 0) may be indicated when the full result is not zero. This occurs when the stored portion of the result is zero, but unstored high-order digits are not zero.

Except for the CVDA and CVAD instructions, a negative zero result is not stored.

When executed by the HP 30012A Expanded Instruction Set, the decimal arithmetic instructions cannot be interrupted. However, when these instructions are performed by the simulation procedures, interrupts are recognized in the manner established for the instructions which make up each procedure.

The ADDD, CMPD, MPYD, NSLD, SLD, SRD, and SUBD instructions can specify overlapping operand fields provided the two signs coincide (share the same byte address).

LANGUAGE EXTENSION INSTRUCTIONS

SECTION IV

This section describes the language extension instructions which complement the basic instruction set of the HP 3000 Computer Systems. Instruction Commentary 1 which follows immediately after the instruction description explains the possible traps for the language extension instructions. Instruction Commentary 2 provides additional information for the CMPS and CMPT instructions. Instruction Commentary 3 provides examples of the EDIT subprograms.

PROGRAM CONTROL INSTRUCTIONS

XBR External branch

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	1	1	1	1	0	0

Control is transferred unconditionally to the location pointed to by the evaluation of the two word label taken from the top of stack. The format for the label is as follows:

S-1	PB rel. address
S-0	SEG#

Both parameters are deleted from the stack.

Traps: Stack Underflow, Bounds Violation, CST Violation Priv. Mode Violation, Absense, Trace

Indicators: Unaffected

PARC Paragraph procedure call

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	1	1	1	1	0	1

Control is transferred to the location pointed to by the evaluation of the two word label at S-2, S-1. The paragraph number given at S-0 is not modified and is used by the END OF PARAGRAPH instruction. The two word label is replaced with a PB relative return address in S-2 and a copy of the status register in S-1. The status register is used to obtain the current segment number. The format for the stack before and after is as follows:

before

S-2	PB rel. address
S-1	SEG#
S-0	paragraph #

after

S-2	PB rel. rtn adar
S-1	STATUS
S-0	paragraph #

None of the parameters are deleted from the stack.

Traps: Stack Overflow, Stack Underflow, Bounds Violation, CST Violation, Priv. Mode Violation, Absence, Trace

Indicators: Unaffected

ENDP End of paragraph

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	1	1	1	1	1	0

The current paragraph number contained in S-0 is compared to the paragraph number of the terminating paragraph in S-1 (which was left by the PARC instruction). If the two paragraph numbers are equal the two words S-3, S-2 which contain PB relative return address and status are used to return from the call made by the PARC instruction. If the two paragraph numbers are not equal the exit is not taken. The format for the stack prior to execution is as follows:

S-3	PB rel. address	
S-2	SEG#	left by PARC inst.
S-1	paragraph #	
S-0	paragraph #	

If (S-0) equals (S-1) then all the parameters are deleted from the stack else only S-0 is deleted.

The ENDP instruction requires the presence of a paragraph number on the stack. The paragraph number is left on the stack by the execution of the PARC instruction. At the entry of a main or subprogram a dummy paragraph number must be placed on the stack. This dummy paragraph number should be an illegal number (such as -1). The entire three word stack marker is not required since no exit will ever be executed using it.

Traps: Stack Underflow, Bounds Violation, CST Violation, Priv. Mode Violation, Absence, Trace

Indicators: Unaffected

EDITED AND NONEDITED MOVE INSTRUCTIONS

EDIT The EDIT instruction moves a string of characters from the source buffer to the target buffer under the control of an EDIT subprogram.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	1	1	1	0	0	y
													0	0	y

When Y = 0, the location is in the PB area.
When Y = 1, the location is in the DB area.

The EDIT instruction, prior to its execution, requires the condition code to be set to reflect the sign of the number being processed for numeric editing. A second indicator, the significance trigger, is maintained by the EDIT instruction, and is set to a 1 when the first non-zero digit is encountered. These indicators control leading zero suppression and replacement, sign insertion, and sign overpunch. Three 8-bit values are maintained by the EDIT instruction and hold the definitions of the fill character, the float character, and loop count. The defaults are as follows:

INDICATOR or DATA ITEM	DEFAULT
Significance Trigger	0
Fill Character	SPACE
Float Character	"s"
Loop Count	0 (equiv. to 256)

The EDIT instruction also maintains three pointers. The three pointers are the Source Pointer, the Target Pointer, and the Operation Code Pointer. These pointers point to the current byte in progress for the respective areas. At the beginning of an EDIT instruction these pointers are set to the values contained in the three words on top of stack. The three buffers (source, target, and subprogram) may overlap in any way desired. When overlap occurs it is possible to get unpredictable results or encounter an error condition by modifying either the source or the subprogram. The stack prior to execution is as follows:

S-3	EDIT subprogram address	
S-2	target address	
S-1	source address	
S-0	ZERO	Interrupt Restart Word

The EDIT instruction is interruptable after each subprogram instruction, and will continue from the point of interruption when control is returned to the instruction. The zero word on top of the stack allows restarting the instruction in the middle of an EDIT sub-program. On completion of the sub-program all four parameters are popped from the stack. The Condition Code and Carry bits are not modified. Overflow may be set. (See Instruction Commentary 1.)

The EDIT subprogram can perform a variety of operations including leading zero suppression and replacement, leading or trailing insertion of the sign, leading or trailing sign overpunch, floating character insertion, punctuation control, and text insertion.

The EDIT subprogram is made up of 8-bit instructions followed by zero or more 8-bit operands. Instructions are included for edited moves, character or sign insertion, pointer modfication, setting and testing the significance trigger and looping. The EDIT subprogram is processed sequentially unless instructed to do otherwise. The op-code pointer is updated after each operation to point to the next sequential op-code. The EDIT instruction will continue to process op-codes until directed to stop by the terminate edit (TE) op-code or an error condition is detected. The EDIT subprogram is located either in the PB or DB relative area.

The 8 bit instructions are divided into two 4-bit fields as follows:

	op-code	immediate operand
bits	0 1 2 3	4 5 6 7

Table 4-1. EDIT Instruction Set Summary

4-BIT OPCODE	MNEMONIC	INSTRUCTION DESCRIPTION
0	MC n	Move n characters
1	MA n	Move n alphabetics
2	MN n	Move n numerics
3	MNS n	Move n numerics suppressed
4	MFL n	Move n numerics with floating insertion
5	IC n,x	Insert character "x" n times
6	ICS n,x	Insert character "x" n times suppressed
7	ICI n,x	Insert n characters immediate
%10	ICSI n,x	Insert n characters suppressed immediate
%11	BRIS d	Branch d bytes if significance equals 1
%12	SUFT d	Subtract d from target pointer
%13	SUFS d	Subtract d from source pointer
%14	ICP m	Insert character punctuation
%15	ICPS m	Insert character punctuation suppressed
%16	IS m	Insert m characters depending on sign
%17	****	This opcode is subdecoded to provide those instructions which require no immediate operand. The operand field is decoded to provide 1 of 16 functions as described in table 4-2.

Table 4-2. EDIT Instruction Set Summary For Opcode = % 17

4-BIT OPCODE	MNEMONIC	INSTRUCTION DESCRIPTION
0	TE	Terminate EDIT
1	ENDF	END floating insertion
2	SST1	Set significance to 1
3	SSTO	Set significance to 0
4	MDWO	Move digit with overpunch
5	SFC x	Set fill character equal to x
6	SFLC x,y	Set float character depending on sign
7	DFLC x,y	Define float character depending on sign
%10	SETC n	Set loop count to n
%11	DBNZ d	Decr. loop count and branch if non zero

EDIT Instruction Immediate Operands

All of the instructions defined by op-codes in the range of 0 to %16 contain at least one operand. The maximum value for the immediate operands is 15. To allow operands outside the range of 1 to 15 the immediate operand may be set to zero and the next sequential byte is used as the operand. This is allowed for only those instructions whose op-codes are in range of 0 to %13. The interpretation of the extended operand is as follows:

op-code range			extended operand range		
min.		max.	min.		max.
0	to	%10	0	to	255
%11	to	%13	-128	to	127

EDIT SUBINSTRUCTIONS

Following is a description of each of the EDIT subinstructions which may be used to construct an EDIT subprogram. (For examples, see Instruction Commentary 3.)

FORMAT TYPE #1 (for instructions: MC,MA,MN,MNS,MFL).
The instruction format for the instructions with only immediate operands is as follows:

0 3	4 7	0 7
opcode	immediate operand	extended operand (optional)
byte 1		byte 2

Note: The immediate (or extended) operand indicates the number of characters to be moved from the source buffer to the target buffer.

MC

Move Characters (format # 1). The MC instruction transfers a specified number of bytes from the source buffer to the target buffer. The immediate (or extended) operand defines a positive byte count. Both the source pointer and the target pointer are increased by the byte count.

Traps: Bounds Violation
Indicators: None

0			3	4 7
0	0	0	0	immediate operand

MA

Move Alphabetic (format # 1). The MA instruction transfers a specified number of alphabetic characters (A-Z, a-z, & SPACE) from the source buffer to the target buffer. The immediate (or extended) operand defines a positive byte count. Both the source pointer and the target pointer are increased by the byte count.

Traps: Bounds Violation, Invalid Alphabetic Character
Indicators: Overflow

0			3	4 7
0	0	0	1	immediate operand

MN

Move Numerics (format # 1). The MN instruction transfers a specified number of numeric characters (0-9 & leading SPACE) from the source buffer to the target buffer. When the first non-zero digit is encountered, the significance trigger is set to 1. The immediate (or extended) operand defines a positive byte count. Both the source pointer and the target pointer are increased by the byte count.

Traps: Bounds Violation, Invalid ASCII Digit
Indicators: Overflow

0			3	4 7
0	0	1	0	immediate operand

MNS

Move Numerics With Zero Supression (format # 1). The MNS instruction transfers a specified number of numeric characters (0-9 & leading SPACE) from the source buffer to the target. While the significance trigger is 0 all zeros and spaces are replaced with the fill character. When the first non-zero digit is encountered the significance trigger is set to a 1. The immediate (or extended) operand defines a positive byte count. Both the source pointer and the target pointer are increased by the byte count.

Traps: Bounds Violation, Invalid ASCII Digit
Indicators: Overflow

0			3	4 7
0	0	1	1	immediate operand

MFL

Move numerics With Floating Insertion (format # 1). The MFL instruction transfers a specified number of numeric characters (0-9 & leading SPACE) from the source buffer to the target. While the significance trigger is 0 all zeros and spaces are replaced with the fill character. When the first non-zero digit is encountered the significance trigger is set to a 1 and the float character is placed in the target buffer followed by the non-zero digit. The immediate (or extended) operand defines a positive byte count. The source pointer is increased by the byte count. The target pointer is increased by the byte count plus one if the significance trigger changes from 0 to 1 else the target pointer is increased by the byte count.

Traps: Bounds Violation, Invalid ASCII Digit

0			3	4 7
0	1	0	0	immediate operand

FORMAT TYPE # 2 (for instructions: IC, ICS). The instruction format for the instructions with immediate operands and a single insertion character is as follows:

0 3	4 7	0 7	0 7
opcode	immediate operand	extended operand (optional)	insert character
byte 1		byte 2	byte 3

Note: If the immediate operand is non-zero then the insert character would appear in byte 2 (not byte 3).

Note: The immediate (or extended) operand indicates the repeat factor to be used for insertion.

IC Insert Character (format # 2). The IC instruction inserts a single character into the target buffer a specified number of times. The immediate (or extended) operand defines a positive repeat count. The character to be inserted is specified in the second or third byte of the IC instruction. The target pointer is increased by the repeat count.
Traps: Bounds Violation
Indicators: None

0			3	4 7
0	1	0	1	immediate operand

ICS Insert Character Suppressed(format # 2). The ICS instruction inserts a single character into the target buffer a specified number of times if the significance trigger is set to a 1. If the significance trigger is set to a 0 the fill character is inserted into the target buffer a specified number of times. The immediate (or extended) operand defines a positive repeat count. The character to be inserted is specified in the second or third byte of the IC instruction. The target pointer is increased by the repeat count.
Traps: Bounds Violation
Indicators: None

0			3	4 7
0	1	1	0	immediate operand

FORMAT TYPE # 3 (for instructions: ICI,ICSI). The instruction format for the instructions with immediate operands and an insertion character string is as follows:

0 3	4 7	0 7	0 7
opcode	immediate operand	extended operand (optional)	character string
byte 1		byte 2	bytes 3 - 257

Note: If the immediate operand is non-zero then the character string would start in byte 2 (not byte 3).

Note: The immediate (or extended) operand indicates the length of the character string to be inserted.

ICI Insert Characters Immediate (format # 3). The ICI instruction inserts a character string of specified length into the target buffer. The immediate (or extended) operand defines a positive character count. The character string to be inserted is specified by a byte array which starts in the second or third byte of the instruction. The target pointer is increased by the character count.
Traps: Bounds Violation
Indicators: None

0–3	4–7
0 1 1 1	immediate operand

ICSI Insert Characters Suppressed Immediate (format # 3). The ICSI instruction inserts a character string of specified length into the target buffer if the significance trigger is set to 1. If it is 0 then the string inserted will be replaced by the fill character. The immediate (or extended) operand defines a positive character count. The character string to be inserted is specified by a byte array which starts in the second or third byte of the instruction. The target pointer is increased by the character count.
Traps: Bounds Violation
Indicators: None

0–3	4–7
1 0 0 0	immediate operand

FORMAT TYPE # 4 (for instructions: BRIS,SUFT,SUFS). The instruction format for the instructions with immediate operands to form displacements to modify pointers is as follows:

0–3	4–7	0–7
opcode	immediate operand	extended operand (optional)
byte 1		byte 2

Note: The immediate operand indicates a positive (1-15) displacement. If it is equal to 0, the extended operand indicates a two's complement displacement.

Note: For BRIS the displacement is added to the opcode pointer. For SUFT and SUFS the displacement is subtracted from the corresponding pointer.

BRIS Branch If Significance Trigger Is Set (format # 4). The BRIS instruction adds a displacement to the opcode pointer if the significance trigger is a 1. The displacement is specified by the immediate (or extended) operand as above. The addition is to the address of the byte containing the displacement (i.e. byte 1 or byte 2).
Traps: Bounds Violation
Indicators: None

0–3	4–7
1 0 0 1	immediate operand

SUFT Subtract From Target Pointer (format # 4). The SUFT instruction subtracts a displacement from the target pointer. The displacement is specified by the immediate (or extended) operand as above. At this point the target pointer is pointing to the next byte to be transferred (stored).
Traps: Bounds Violation
Indicators: None

0–3	4–7
1 0 1 0	immediate operand

SUFS Subtract From Source Pointer (format # 4). The SUFS instruction subtracts a displacement from the source pointer. The displacement is specified by the immediate (or extended) operand as above. At this point the source pointer is pointing to the next byte to be fetched.
Traps: Bounds Violation
Indicators: None

0			3	4 7
1	0	1	1	immediate operand

FORMAT TYPE # 5 (for instructions: ICP,ICPS). The instruction format for the instructions with only immediate operands to generate punctuation characters is as follows:

0 3	4 7
opcode	immediate operand

byte 1

Note: The immediate operand indicates an index into the ASCII character set. The character formed is equal to index+%40.

ICP Insert Character Punctuation (format # 5). The ICP instruction inserts a single character into the target buffer. The immediate operand defines the ASCII character to be inserted. The ASCII character inserted equals the operand plus %40. The target pointer is increased by one.
Traps: Bounds Violation
Indicators: None

0			3	4 7
1	1	0	0	immediate operand

ICPS Insert Character Punctuation Suppressed (format # 5). The ICPS instruction inserts a single character into the target buffer if the significance trigger is set to a 1. If the significance trigger is set to a 0 the fill character is inserted into the target buffer. The immediate operand defines the ASCII character to be inserted. The ASCII character inserted equals the operand plus %40. The target pointer is increased by one.
Traps: Bounds Violation
Indicators: None

0			3	4 7
1	1	0	1	immediate operand

FORMAT TYPE # 6 (for instructions: IS). The instruction format for the insert sign instruction which has one immediate operand and two insertion character strings is as follows:

0 3	4 7	0 7	0 7
opcode	immediate operand	character string if sign is pos.	character string if sign is neg.
byte 1		bytes X0 - X1	bytes X2 - X3

Note: The length of the character strings is given by the immediate operand. Typically this would be 1 or 2. The relations between the immediate operand and the byte positions X0,X1,X2, and X3 are as follows:

X0 equals 2
X1 equals 1+m
X2 equals 2+m
X3 equals 1+m*2

where m is the immediate operand

IS Insert Characters Depending On Sign (format # 6). The IS instruction inserts one of two specified character strings depending on the sign of the source as specified in the condition code. If the sign is positive (= CCG or CCE) the first character string will be inserted else the second character string will be inserted (see format #6 for the byte positions). The immediate operand specifies the length of both character strings. The target pointer is increased by this length.
Traps: Bounds Violation
Indicators: None

0			3	4 – 7
1	1	1	0	immediate operand

FORMAT TYPE # 7 (for instructions: TE,ENDF,SST1,SST0, MDWO). The instruction format for those instructions which are one byte long and have an opcode of %17 is as follows:

0 – 3	4 – 7
%17	secondary opcode

byte 1

Note: The secondary opcode is in the range of 0 to 4.

TE Terminate EDIT (format # 7). The TE instruction terminates the EDIT subprogram and deletes all parameters from the stack.
Traps: Stack Underflow
Indicators: None

0			3	4			7
1	1	1	1	0	0	0	0

ENDF End Floating Insertion (format # 7). The ENDF instruction inserts the float character into the target buffer and increases the target pointer by 1 if the significance trigger is set to a 0. If the significance trigger is set to a 1 then no action is taken.
Traps: Bounds Violation
Indicators: None

0			3	4			7
1	1	1	1	0	0	0	1

SST1 Set Significance Trigger To One (format # 7). The SST1 instruction sets the significance trigger to a 1.
Traps: None
Indicators: None

0			3	4			7
1	1	1	1	0	0	1	0

SST0 Set Significance Trigger To Zero (format # 7). The SST0 instruction sets the significance trigger to a 0.
Traps: None
Indicators: None

0			3	4			7
1	1	1	1	0	0	1	1

MDWO Move Digit With Overpunch (format # 7). The MDWO instruction transfers a single numeric character (0-9 & leading SPACE) from the source buffer to the target

0			3	4			7
1	1	1	1	0	1	0	0

buffer. The actual character placed in the target buffer is a function of the digit and the sign of the number as follows:

Source Digit	Target Digit If Positive	Target Digit If Negative
0 or space	{ or %173	} or %175
1	A or %101	J or %112
2	B or %102	K or %113
3	C or %103	L or %114
4	D or %104	M or %115
5	E or %105	N or %116
6	F or %106	O or %117
7	G or %107	P or %120
8	H or %110	Q or %121
9	I or %111	R or %122

Both the target pointer and the source pointer are increased by one.
Traps: Bounds Violation, Invalid ASCII digit
Indicators: Overflow

FORMAT TYPE # 8 (for instructions: SFC,SFLC,SETC,DBNZ). The instruction format for those instructions which are two bytes long and have an opcode of %17 is as follows:

0 3	4 7	0 7
%17	secondary opcode	operand 1 byte (required)
byte 1		byte 2

Note: The secondary opcode is in the range of 5 to %11.

SFC Set Fill Character (format # 8). The SFC instruction defines the fill character which is used in the MNS, MFL, ICS, ICSI, and ICPS instructions to suppress leading zeros and fixed insertion characters. The second byte of this instruction specifies the ASCII character to define the fill character.
Traps: None
Indicators: None

0			3	4			7
1	1	1	1	0	1	0	1

SFLC Set Float Character (format # 8). The SFLC instruction defines the float character which is used in the MFL instruction to provide floating sign insertion. The second byte of this instruction specifies two characters as two 4 bit operands. The ASCII character which defines the float character is equal to the operand plus %40. The first operand is used if the sign of the source as specified by the condition code is positive (CCG or CCE) else the second operand is used.
Traps: None
Indicators: None

0			3	4			7
1	1	1	1	0	1	1	0

SETC Set Loop Count (format # 8). The SETC instruction defines a loop count. The loop count is defined by the second byte of this instruction. When initialized to 0, the count is equivalent to 256.
Traps: None
Indicators: None

0			3	4			7
1	1	1	1	1	0	0	0

DBNZ Decrement Count, Branch If Non-zero (format # 8). The DBNZ instruction decrements the loop count as defined by the SETC instruction and adds a displacement to the opcode pointer if the resulting value of the loop count is not equal to zero. The displacement is specified by the second byte as a two's complement integer. The branch is relative to the last fetched byte of the sub-program — i.e. to byte 2.
Traps: Bounds Violation
Indicators: None

0			3	4			7
1	1	1	1	1	0	0	1

FORMAT TYPE # 9 (for instructions: DFLC). The instruction format for those instructions which are three bytes long and have an opcode of %17 is as follows:

0 3	4 7	0 7	0 7
%17	secondary opcode	1st operand (required)	2nd operand (required)
byte 1		byte 2	

Note: The secondary opcode is in the range of 5 to %11.

DFLC Define Float Character (format # 9). The DFLC instruction defines the float character which is used in the MFL instruction to provide floating sign insertion. The second and third bytes of this instruction specify two ASCII characters. The first operand is used if the sign of the source as specified by the condition code is positive (CCG or CCE) else the second operand is used.
Traps: None
Indicators: None

0			3	4			7
1	1	1	1	0	1	1	1

CODE CONVERTING AND COLLATING INSTRUCTIONS

TR Translate. The TRANSLATE instruction converts a string of characters from one character set to another character set. The bytes from the source string are used as arguments to reference the translation table. The bytes selected from the table are placed in corresponding positions in the target string. The format of the stack prior to execution is as follows:

S-3	byte address of translation tbl
S-2	byte address of source string
S-1	byte address of target string
S-0	length of source string

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	1	1	1	1	1	1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	Y

When Y = 0 the location is in the PB relative area.
When Y = 1 the location is DB relative.

The source and target string byte addresses are DB-relative. The translation table byte address is PB- or DB-relative. If the byte count = 0, the stack is popped by 4 and execution continues with the next instruction. This instruction is interruptable after each byte transfer. On completion of the instruction all 4 parameters are deleted from the stack.
Traps: Stack overflow, Stack underflow, Bounds Violation
Indicators: Unaffected.

CMPS Compare Strings. The CMPS instruction compares bytes from a DB-relative source string to those of a PB- or DB-relative target string. If the strings are of unequal length the ASCII blank is used as the fill character for the shorter string. The stack prior to execution of the instruction is as follows:

S-3	byte address of target string
S-2	length of target string
S-1	byte address of source string
S-0	length of source string

The instruction terminates when an unequal comparison has been made or the maximum length has been compared. Comparisons are made left-to-right. This instruction is interruptable between each byte comparison. On completion of the instruction, all 4 parameters are popped from the stack. (See Instruction Commentary 2.)
Traps: Stack Overflow, Stack Underflow, Bounds Violation
Indicators: CCE if no mismatch over max. length is found, or if both lengths = 0 on entry
CCG if target byte < source byte
CCL if target byte > source byte

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	1	1	1	0	1	Y

When Y = 0, the location is PB relative; When Y = 1, the location is DB relative.

CMPT Compare Translated Strings. The CMPT instruction compares byte strings using a DB-relative translation table. A DB-relative source byte is used as an index into the table to obtain the first byte to be compared. If the instruction is DB-relative then the DB-relative target byte is used as an index into the same table to obtain the second byte to be compared — otherwise the PB-relative target byte is the second (untranslated) byte for the comparison. If the strings are of unequal length the (always translated) ASCII blank is used as the fill character for the shorter string. The stack prior to execution of the instruction is as follows:

S-4	byte address of translation tbl
S-3	byte address of target string
S-2	length of target string
S-1	byte address of source string
S-0	length of source string

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	1	1	1	1	1	1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	Y

When Y = 0, the location is PB relative; when Y = 1, the location is DB.

The instruction terminates when an unequal comparison has been made or the maximum length has been compared. Comparisons are made left-to-right. This instruction is interruptable between each byte comparison. On completion of the instruction, all 5 parameters are popped from the stack. (See Instruction Commentary 2.)
Traps: Stack Overflow, Stack Underflow, Bounds Violation
Indicators: CCE if no mismatch over the max. length is found, or if both lengths = 0 on entry
CCG if target byte < source byte
CCL if target byte > source byte

NUMERIC CONVERSION AND LOAD INSTRUCTIONS

ALGN Align Numeric. The ALGN instruction transfers a numeric item from the DB-relative source buffer to the DB-relative target buffer. The transfer aligns the source item to the target item by decimal point. The lengths and the number of digits to the right of the decimal point are bytes. The stack before execution is as follows:

S-3	Target Byte Address	
S-2	F1 L1	(target)
S-1	Source Byte Address	
S-0	F2 L2	(source)

L1 and L2 specify the length of the data item for the target and the source. F1 and F2 specify the number of digits to the right of the decimal point for the target and the source. L1, L2, F1, and F2 are restricted as follows:

F1 <= L1 <= 28 and F2 <= L2 <= 28

The alignment will be performed as follows:

1) The source will be truncated as necessary as defined by the target (i.e. at either end — however non-transferred characters will still be validated).
2) Leading Blanks in the source will be treated as zeros (converted to 0's in target if transferred).
3) The target will be zero filled.
4) The source may contain only numeric characters (0-9 & leading space) and trailing overpunch.
5) If the last char. of the source is overpunched, then the last char. of the target will be overpunched.

The SDEC bit allows leaving either the target address on the stack or deleting all parameters. If the source or target character count is zero the SDEC operation is performed and execution continues with the next instruction.
SDEC = 0, delete all parameters except target address
SDEC = 1, delete all 4 parameters
Traps: Invalid ASCII Digit, Invalid Operand Length Stack Overflow, Stack Underflow, Bounds Violation
Indicators: Overflow

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	1	1	0	0	0	S

Where S = SDEC.

CVND Convert Numeric Display. The CVND instruction produces a default numeric display item (a target string) from a signed or unsigned numeric display item (a source string). The default numeric display item is a string of ASCII numeric characters where the low order digit may be a numeric character or an overpunch character. Three bits within the instruction specify if the source is signed or unsigned and the location of the sign for signed items. The combinations are as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	1	1	1	1	1	1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	1	X	X	X	S

Note: Bits 11-15 of the 2nd op range from %20 to %37
xxx = Sign Control Field (as above)
S = SDEC field

0	0	0	sign is leading separate
0	0	1	sign is trailing separate
0	1	0	sign is leading overpunch
0	1	1	sign is trailing overpunch or unsigned
1	X	X	source is unsigned

The source item may contain characters from each of the following groups of characters:

numeric (0-9 & leading SPACE)
sign (SPACE, +, −)
overpunch ({, A, B, C, D, E, F, G, H, I,
}, J, K, L, M, N, O, P, Q, R,)

Note: sign is leading overpunch may not contain leading blanks.

The overpunch characters are described in the EDIT instruction description (see MDWO instruction). Leading spaces are converted to 0's. For the first four cases, the target string will have its last character overpunched in accordance with the source sign (NOT the condition code). The last case will yield an unsigned target string. The SDEC bit allows leaving either the target address on the stack or deleting all parameters. If the source character count is zero the SDEC operation is performed and execution continues with the next instruction. Byte addresses below are DB-relative.

The stack before execution is as follows:

S-2	Target byte Address
S-1	Source byte Address
S-0	Source Character Count

SDEC = 0, delete both source parameters
SDEC = 1, delete both parameters
Traps: Invalid ASCII Digit
Invalid Source Character Count
Stack Overflow, Stack Underflow, Bounds Violation
Indicators: Overflow

ABSN Absolute Numeric. The ABSN instruction produces an unsigned numeric display item from a default numeric display item and sets the condition code to reflect the sign of the source. The default numeric display item is a string of ASCII numeric characters where the low order

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	1	1	0	0	1	S

Where S = SDEC

digit may be a numeric character or an overpunch character. The source item may contain characters from each of the following groups of characters:

numeric (0-9 & leading SPACE)
overpunch ({, A, B, C, D, E, F, G, H, I,
}, J, K, L, M, N, O, P, Q, R,)

The overpunch characters are described in the EDIT instruction description (see MDWO instruction). Leading spaces are converted to 0's. The SDEC bit allows leaving either the source address on the stack or deleting all parameters. If the source character count is zero the SDEC operation is performed and execution continues with the next instruction.

The stack before execution is as follows:

S-1	Source byte Address
S-0	Source Character Count

SDEC = 0, delete count
SDEC = 1, delete both parameters
Traps: Invalid ASCII Digit
Invalid Source Character Count
Stack Overflow, Stack Underflow, Bounds Violation
Indicators: Overflow

ABSD Absolute Decimal. The ABSD instruction changes the sign of a packed decimal value to %17. This produces an unsigned packed decimal result. The SDEC bit allows leaving either the target address on the stack or deleting both parameters. The stack before execution is as follows:

S-1	Byte Address
S-0	Digit Count

SDEC = 0, delete digit count parameter
SDEC = 1, delete both parameters
Traps: Invalid digit count
Stack Overflow, Stack Underflow, Bounds Violation
Indicators: CCA on original source, Overflow

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	1	1	1	1	1	1

0	1	2	3	4	5	6	7	8	9	10	11	.2	13	14	15
0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	S

Where S = SDEC

NEGD Negate Decimal. The NEGD instruction negates a packed decimal value. The SDEC bit allows leaving either the target address on the stack or deleting both parameters. The stack before execution is as follows:

S-1	Byte Address
S-0	Digit Count

SDEC = 0, delete digit count parameter
SDEC = 1, delete both parameters
Traps: Invalid digit count
Stack Overflow, Stack Underflow, Bounds Violation
Indicators: CCA on result, Overflow

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	1	1	1	1	1	1

0	1	2	3	4	5	8	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	S

Where S = SDEC

LDW Load Word (2 Consecutive Bytes). The LDW instruction loads two bytes to the TOS from the DB relative address on TOS. The SDEC bit allows leaving or deleting the address on the stack.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	1	1	1	1	1	1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	S

Where S = SDEC

The stack before execution is as follows:

S-0	DB relative byte address

SDEC = 0, leaves address on the stack
SDEC = 1, deletes address from stack
Traps: Stack Overflow, Stack Underflow, Bounds Violation
Indicators: Unaffected

LDDW Load Double Word (4 Consecutive Bytes). The LDDW instruction loads four bytes to the TOS from the DB relative address on TOS. The SDEC bit allows leaving or deleting the address on the stack.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	0	0	0	0	1	0	0	1	1	1	1	1	1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	S

Where S = SDEC

The stack before execution is as follows:

S-0	DB relative byte address

SDEC = 0, leaves address on the stack
SDEC = 1, deletes address from stack
Traps: Stack Overflow, Stack Underflow, Bounds Violation
Indicators: Unaffected

INSTRUCTION COMMENTARY

1 This commentary explains the traps for the Language Extension Instructions. For these instructions a variety of traps is possible.

If an invalid count or an invalid ASCII digit is detected, the instruction terminates at the point of error, and one of two possibilities occurs (note: check "Indicators" for a particular instruction to see if this type of trap can occur). If the User Traps bit (STA(2)) is not set, the Overflow bit (STA(4)) is set, the stack is popped in accordance with the instruction, and execution continues with the following instruction. If the Traps bit is set, the Overflow bit is not set, the stack is not popped, and a trap to the Traps segment, segment 1 is taken.

For the cases of Stack Overflow, Stack Underflow, Bounds Violation, CST Violation, and Privileged Mode Violation, the instruction is aborted at the point of error and an unconditional trap is taken to Segment 1. In general, the state of the stack is not known in these cases. These traps are handled the same as for the basic machine instruction set.

Unimplemented instructions also trap to segment 1 as in the basic instruction set. The X-register will contain the contents of the CIR-register. This is the actual unimplemented opcode in the case of single word instructions, or %020477 in the case of double word instructions. In either case, the delta P value in the stack marker, when added to PB, points to the next true instruction (i.e. not to the second word of a double word instruction).

2 The following commentary provides a further explanation of the comparison instructions CMPS and CMPT.

There are four types of alphanumeric comparisons which are described in the table below:

Case #	Translation Required	Filling Required	Equal Length	Target Location
1	none	none	equal	DB+/PB+
2	none	blanks	no	DB+/PB+
3	yes (source)	blanks	no	PB+
4	yes (both)	blanks	no	DB+

Note: source location and translation table are always DB-relative. Target location may be either PB- or DB-relative. Note also that the Condition Codes CCG and CCL are reversed from that of the CMPB instruction, and the PB indicator applies to the target instead of the source.

CASE #1. This case compares two strings of equal length and provides no translation of operands. This can be implemented using the CMPB instruction.

CASE #2 CMPS Compare Strings. This case compares two strings of different length and provides no translation of operands. The ASCII blank character is used as the fill character for the shorter string.

CASE #3 CMPT Compare Translated Strings. This case compares two strings of different length while converting the source string using a translation table (see TRANSLATE instruction). The translation of the ASCII blank is used as the fill character for the shorter string, regardless of whether it is the source or target string.

CASE #4 CMPT Compare Translated Strings. This case compares two strings of different length while converting both strings using a translation table (see TRANSLATE instruction). The translation of the ASCII blank is used as the fill character for the shorter string.

3 The following commentary provides various examples of EDIT subprograms.

PICTURE	SIGN	SOURCE	TARGET
*******	****	******	******

EXAMPLE #1

PICTURE	SIGN	SOURCE	TARGET
9999.99	+	000123	0001.23

EDIT subprogram:

MN 4	ICP "."	MN 2	TE

EXAMPLE #2

PICTURE	SIGN	SOURCE	TARGET
****.99	+	001234	**12.34

EDIT subprogram:

SFC	"*"	MNS 4	ICP "."	MN 2
TE				

PICTURE	SIGN	SOURCE	TARGET
* * * * * * *	* * * *	* * * * * *	* * * * * *

EXAMPLE #7

PICTURE	SIGN	SOURCE	TARGET
– – – 9 . 9 9	–	0 0 1 2 3	– 1 . 2 3
			. . . .

EDIT subprogram:

SFLC	" ","–"	MFL 2	ENDF	MN 1
ICP "."	MN 2	TE		

EXAMPLE #8

PICTURE	SIGN	SOURCE	TARGET
* * * * . * *	+	0 0 0 0 0	* * * * . * *
			. . . .

EDIT subprogram:

SFC	"*"	MNS 4	ICP "."	MN 2
BRIS 4	SUFT 2	IC 2	"*"	TE

EXAMPLE #9

PICTURE	SIGN	SOURCE	TARGET
Z Z Z Z . Z Z	+	0 0 0 0 0 1	. 0 1
			. . . .

EDIT subprogram:

MNS 4	ICP "."	MN 2	BRIS 4	SUFT 3
IC 3	" "	TE		

EXAMPLE #10

PICTURE	SIGN	SOURCE	TARGET
+ + + + +	+	0 0 0 0	
			. . . . .

EDIT subprogram:

SFLC	"+","–"	MFL 2	BRIS 2	ICP " "
TE				

	PICTURE	SIGN	SOURCE	TARGET
	*******	****	******	******
EXAMPLE #11				
	AAABAA		ABCDE	ABC DE
				......

EDIT subprogram:

MA 3	ICP " "	MA 2	TE

EXAMPLE #12

$999CR	–	012	$012CR
			......

EDIT subprogram:

ICP "$"	MN 3	IS 2	" "	" "
"C"	"R"	TE		

EXAMPLE #13

AX9AX9		! " # $ % &	! " # $ % &
			......

EDIT subprogram:

MC 6	TE

EXAMPLE #14

+$$$V$$	–	0123	– $123
			......

EDIT subprogram:

IS 1	"+"	"–"	MFL 2	ENDF
MN 2	BRIS 4	SUFT 7	IC 7	" "
TE				

Table A-1. Alphabetical Listing of Instructions

ABSO	4-15	DABZ	2-15	FIXR	2-4	LSL	2-10	QASL	2-12	SXIT	2-28	EIS	
ABSN	4-14	DADD	2-2	FIXT	2-5	LSR	2-10	QASR	2-13				
ADAX	2-7	DASL	2-11	FLT	2-3	LST	2-23					ADDD	3-5
ADBX	2-7	DASR	2-11	FMPY	2-4	LSUB	2-5					CMPD	3-6
ADD	2-1	DCMP	2-3	FNEG	2-4							CVAD	3-3
ADDI	2-24	DCSL	2-11	FSUB	2-4					TASL	2-12	CVBD	3-4
ADDM	2-41	DCSR	2-12					RCCR	2-31	TASR	2-12	CVDA	3-4
ADDS	2-27	DDEL	2-9			MABS	2-20	RCLK	2-27	TBA	2-38	CVDB	3-5
ADXA	2-7	DDIV	2-3	HALT	2-30	MCS	2-36	RIO	2-34	TBC	2-13	DMPY	3-8
ADXB	2-7	DDUP	2-9	HIOP	2-37	MDS	2-22	RIOC	2-37	TBX	2-38		
ADXI	2-25	DECA	2-6	IABZ	2-14	MFDS	2-21	RMSK	2-33	TCBC	2-14	EADD	3-1
ALGN	4-13	DECB	2-7	INCA	2-6	MOVE	2-17	RSW	2-31	TEST	2-9	ECMP	3-2
AND	2-6	DECM	2-41	INCB	2-6	MPY	2-1			TIO	2-34	EDIV	3-2
ANDI	2-26	DECX	2-6	INCM	2-41	MPYI	2-24			TNSL	2-12	EMPY	3-2
ASL	2-10	DEL	2-9	INCX	2-6	MPYL	2-2			TOFF	2-32	ENEG	3-2
ASR	2-10	DELB	2-9	INIT	2-36	MPYM	2-41			TON	2-32	ESUB	3-1
		DFLT	2-3	IXBZ	2-14	MTBA	2-38	SBXI	2-25	TR	4-11		
		DISP	2-29	IXIT	2-29	MTBX	2-38	SCAL	2-28	TRBC	2-13	MPYD	3-8
		DIV	2-1			MTDS	2-21	SCAN	2-13	TSBC	2-13	NSLD	3-7
BCC	2-16	DIVI	2-24			MVB	2-17	SCLK	2-27			SLD	3-6
BCY	2-15	DIVL	2-2	LADD	2-5	MVBL	2-19	SCLR	2-32			SRD	3-7
BNCY	2-15	DLSL	2-11	LCMP	2-5	MVBW	2-18	SCU	2-19			SUBD	3-6
BNOV	2-15	DLSR	2-11	LDB	2-40	MVLB	2-20	SCW	2-19	UNLK	2-30		
BOV	2-15	DMUL	2-3	LDD	2-39			SDEA	2-23				
BR	2-16	DNEG	2-3	LDDW	4-16			SED	2-33				
BRE	2-15	DPF	2-4	LDEA	2-23	NEG	2-2	SEML	2-37				
BRO	2-15	DSUB	2-2	LDI	2-24	NEGD	4-15	SETR	2-26	WIO	2-34		
BTST	2-9	DTST	2-9	LDIV	2-5	NOP	2-10	SIN	2-35	WIOC	2-35		
		DUMP	2-35	LDNI	2-25	NOT	2-6	SINC	2-32				
		DUP	2-9	LDPN	2-39			SIO	2-33				
CAB	2-8	DXBZ	2-14	LDPP	2-36			SIOP	2-36				
CIO	2-34	DXCH	2-8	LDW	4-16	OR	2-6	SMSK	2-33	XAX	2-8		
CMD	2-34	DZRO	2-8	LDX	2-39	ORI	2-25	SSEA	2-23	XBR	4-1		
CMP	2-2			LDXA	2-7			SST	2-23	XBX	2-8		
CMPB	2-18			LDXB	2-7			STAX	2-7	XCH	2-8		
CMPI	2-24	EDIT	4-2	LDXI	2-24	PARC	4-1	STB	2-40	XCHD	2-27		
CMPM	2-41	ENDP	4-2	LDXN	2-25	PAUS	2-30	STBX	2-7	XEQ	2-31		
CMPN	2-25	EXF	2-14	LLBL	2-29	PCAL	2-28	STD	2-40	XOR	2-6		
CMPS	4-12	EXIT	2-28	LLSH	2-31	PCN	2-31	STOR	2-36	XORI	2-25		
CMPT	4-12			LMPY	2-5	PLDA	2-22	STRT	2-37				
CPRB	2-16			LOAD	2-39	PSDB	2-29	SUB	2-1				
CSL	2-10	FADD	2-4	*LOCK	2-30	PSEB	2-30	SUBI	2-24	ZERO	2-8		
CSR	2-11	FCMP	2-3	LRA	2-40	PSHR	2-26	SUBM	2-41	ZROB	2-8		
CVND	4-14	FDIV	2-4	LSEA	2-23	PSTA	2-22	SUBS	2-27	ZROX	2-8		

Part No. 30000-90022
Index No. 3GENL.320.30000-90022
Printed in U.S.A. 2/80

HEWLETT PACKARD

Sales and service from 172 offices in 65 countries.
5303 Stevens Creek Blvd., Santa Clara, California 95050